Université de Montréal

Apprentissage de représentations sur-complètes par entraînement d'auto-encodeurs

par
Isabelle Lajoie

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Décembre, 2009

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Apprentissage de représentations sur-complètes par entraînement d'auto-encodeurs

présenté par:

Isabelle Lajoie

a été évalué par un jury composé des personnes suivantes:

Alain Tapp
président-rapporteur

Pascal Vincent
directeur de recherche

Yoshua Bengio
membre du jury

RÉSUMÉ

Les avancés dans le domaine de l'intelligence artificielle, permettent à des systèmes informatiques de résoudre des tâches de plus en plus complexes liées par exemple à la vision, à la compréhension de signaux sonores ou au traitement de la langue. Parmi les modèles existants, on retrouve les Réseaux de Neurones Artificiels (RNA), dont la popularité a fait un grand bond en avant avec la découverte de Hinton et al. [22], soit l'utilisation de Machines de Boltzmann Restreintes (RBM) pour un pré-entraînement non-supervisé couche après couche, facilitant grandement l'entraînement supervisé du réseau à plusieurs couches cachées (DBN), entraînement qui s'avérait jusqu'alors très difficile à réussir. Depuis cette découverte, des chercheurs ont étudié l'efficacité de nouvelles stratégies de pré-entraînement, telles que l'empilement d'auto-encodeurs traditionnels (SAE) [5, 38], et l'empilement d'auto-encodeur débruiteur (SDAE) [44].

C'est dans ce contexte qu'a débuté la présente étude. Après un bref passage en revue des notions de base du domaine de l'apprentissage machine et des méthodes de pré-entraînement employées jusqu'à présent avec les modules RBM, AE et DAE, nous avons approfondi notre compréhension du pré-entraînement de type SDAE, exploré ses différentes propriétés et étudié des variantes de SDAE comme stratégie d'initialisation d'architecture profonde. Nous avons ainsi pu, entre autres choses, mettre en lumière l'influence du niveau de bruit, du nombre de couches et du nombre d'unités cachées sur l'erreur de généralisation du SDAE. Nous avons constaté une amélioration de la performance sur la tâche supervisée avec l'utilisation des bruits *poivre et sel* (PS) et *gaussien* (GS), bruits s'avérant mieux justifiés que celui utilisé jusqu'à présent, soit le *masque à zéro* (MN). De plus, nous avons démontré que la performance profitait d'une emphase imposée sur la reconstruction des données corrompues durant l'entraînement des différents DAE. Nos travaux ont aussi permis de révéler que le DAE était en mesure d'apprendre, sur des images naturelles, des filtres semblables à ceux retrouvés dans les cellules V1 du cortex visuel, soit des filtres détecteurs de bordures. Nous aurons par ailleurs pu montrer que les représentations apprises du SDAE, composées des caractéristiques ainsi extraites, s'avéraient fort utiles à l'apprentissage d'une machine à vecteurs de

support (SVM) linéaire ou à noyau gaussien, améliorant grandement sa performance de généralisation. Aussi, nous aurons observé que similairement au DBN, et contrairement au SAE, le SDAE possédait une bonne capacité en tant que modèle générateur. Nous avons également ouvert la porte à de nouvelles stratégies de pré-entraînement et découvert le potentiel de l'une d'entre elles, soit l'empilement d'auto-encodeurs rebruiteurs (SRAE).

Mots clés: réseau de neurones artificiel, architecture profonde, apprentissage non-supervisé, auto-encodeur débruiteur, machine de Boltzmann restreinte

ABSTRACT

Progress in the machine learning domain allows computational system to address more and more complex tasks associated with vision, audio signal or natural language processing. Among the existing models, we find the Artificial Neural Network (ANN), whose popularity increased suddenly with the recent breakthrough of Hinton et al. [22], that consists in using Restricted Boltzmann Machines (RBM) for performing an unsupervised, layer by layer, pre-training initialization, of a Deep Belief Network (DBN), which enables the subsequent successful supervised training of such architecture. Since this discovery, researchers studied the efficiency of other similar pre-training strategies such as the stacking of traditional auto-encoder (SAE) [5, 38] and the stacking of denoising auto-encoder (SDAE) [44].

This is the context in which the present study started. After a brief introduction of the basic machine learning principles and of the pre-training methods used until now with RBM, AE and DAE modules, we performed a series of experiments to deepen our understanding of pre-training with SDAE, explored its different proprieties and explored variations on the DAE algorithm as alternative strategies to initialize deep networks. We evaluated the sensitivity to the noise level, and influence of number of layers and number of hidden units on the generalization error obtained with SDAE. We experimented with other noise types and saw improved performance on the supervised task with the use of *pepper and salt* noise (PS) or *gaussian* noise (GS), noise types that are more justified then the one used until now which is *masking noise* (MN). Moreover, modifying the algorithm by imposing an emphasis on the corrupted components reconstruction during the unsupervised training of each different DAE showed encouraging performance improvements. Our work also allowed to reveal that DAE was capable of learning, on naturals images, filters similar to those found in V1 cells of the visual cortex, that are in essence edges detectors. In addition, we were able to verify that the learned representations of SDAE, are very good characteristics to be fed to a linear or gaussian support vector machine (SVM), considerably enhancing its generalization performance. Also, we observed that, alike DBN, and unlike SAE, the SDAE had the potential to be used

as a good generative model. As well, we opened the door to novel pre-training strategies and discovered the potential of one of them : the stacking of renoising auto-encoders (SRAE).

Keywords : neural network, deep architecture, unsupervised learning, denoising autoencoder, restricted Boltzmann machine

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES

ANN	Artificial Neural Network
DBN	Deep Belief Network
I.I.D	Independant et Identiquement Distribué
MLP	Multi-Layer Perceptron
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine
SIFT	Scale Invariant Feature Transform
SVM	Support Vector Machines
AE	Auto-Encoder
DAE	Denoising Auto-Encoder
SDAE	Stacked Denoising Auto-Encoder
MAE	"deMissing" Auto-Encoder
MD	Missing Data
SMAE	Stacked "deMissing" Auto-Encoder
RAE	Renoising Auto-Encoder
SRAE	Stacked Renoising Auto-Encoder
MN	Masking Noise
PS	Pepper Salt Noise
GS	Gaussian Noise

NOTATION

$\mathscr{D}$	Ensemble contenant les exemples d'entraînement et possiblement les cibles.
$\mathscr{D}_{\text{valid}}$	Ensemble contenant les exemples $\notin \mathscr{D}$: utilisés pour choisir les hyper-paramètres optimaux θ_H
$\mathscr{D}_{\text{test}}$	Ensemble contenant les examples $\notin \mathscr{D}, \mathscr{D}_{\text{valid}}$: utilisé pour estimer l'erreur de généralisation
X	Variables aléatoire
Y	Variables aléatoire
$p(x)$	Model estimant la vraie distribution empirique $p_T(x)$
$p(x,y)$	Model estimant la vraie jointe probabilité $p_T(x,y)$
$p(y\|x)$	Model estimant la vraie probabilité conditionnelle $p_T(y\|x)$
$E_{p(x)}$	Espérance de la densité de probabilité $p(x)$
d	Dimension de la couche d'entrée d'un DBN
d_h	Dimension de la couche cachée d'un DBN
m	Dimension de la couche de sortie d'un DBN
$h^{(a)}, h$	Vecteur des unités cachées d'un DBN, avant et après la fonction d'activation
$o^{(a)}, o$	Vecteur des unités de sortie d'un DBN, avant et après la fonction d'activation
(c, V, b, W)	Paramètres d'un RBM ou d'un AE : vecteur de biais b et matrice de poids W de la couche cachée ; vecteur de biais c et matrice de poids V de la couche de sortie
ϕ	Fonction non-linéaire
θ	Ensemble de paramètre à apprendre
f	Fonction de prédiction
f_θ	Fonction f paramétrisée par les paramètres θ
$g(x)$	Fonction discriminante
$\mathscr{L}$	Fonction de perte à minimiser
$\mathscr{R}(f, \mathscr{D})$	Risque empirique de la fonction de prédiction f mesurée sur l'ensemble d'entraînement $\mathscr{D}$

$\mathcal{N}_{\mu,\sigma}$	Distribution gaussienne avec moyenne μ et covariance σ
$\frac{\partial f}{\partial x}$	dérivé de la fonction f par rapport à x
$\mathbb{R}^d$	Ensemble de tout les nombres réels de dimension d
W	L'usage d'une majuscule spécifie une matrice, à moins d'indication contraire
w	L'usage d'un minuscule spécifie un vecteur, à moins d'indication contraire
W^T	Transposer d'une matrice ou d'un vecteur
$W_{i.}$	i-ième rangée de la matrice W
$W_{.j}$	j-ième colonne de la matrice W
W_{ij}	Élément de la i-ième rangée et j-ième colonne de la matrice W
w_i	i-ième élément du vecteur w
$x^{(i)}$	i-ième exemple d'entraînement, $x^{(i)} \in \mathbb{R}^d$
$x \sim p(x)$	x est un échantillon tiré de la densité de probabilité $p(x)$
$y^{(i)}$	Cible pour le i-ième exemple d'entraînement
$t^{(i)}$	Prédiction pour le i-ième exemple d'entraînement
$z^{(i)}$	En apprentissage supervisé, équivaut au couple $(x^{(i)}, y^{(i)})$; en apprentissage non-supervisé, équivaut à $(x^{(i)})$

À mes parents qui ont toujours su m'appuyer et me conseiller
dans mes diverses démarches.

REMERCIEMENTS

Un merci tout particulier à Pascal Vincent, pour avoir cru en moi et m'avoir donné ma chance dans le domaine de l'apprentissage automatique. Merci pour ton encadrement, ton enseignement et ta disponibilité, et ce, malgré une charge de temps bien remplie. Ce fut un réel plaisir de travailler à tes côtés.

Merci à mes collègues du LISA et du GAMME pour votre (oh combien) agréable compagnie, et pour m'avoir partagé votre passion de la recherche. Un merci spécial à Hugo Larochelle pour m'avoir généreusement transmis les outils et connaissances nécessaires dès mes débuts et aux moments opportuns. Merci à Fréderic Bastien pour ton aide constante et à plusieurs niveaux.

Merci à ma famille, spécialement à mes parents, pour votre présence, vos encouragements et votre confiance en moi. Merci, parce que vous comprenez mon horaire de temps bien souvent trop chargé, et parce que vous êtes tout simplement merveilleux.

Finalement, à mes cher(e)s ami(e)s qui agrémentez mon quotidien, qui partagez mes joies et mes peines, merci infiniement.

CHAPITRE 1

INTRODUCTION

Le présent chapitre permettra de mettre en lumière les notions de base associées au domaine de l'apprentissage automatique, ses défis principaux et la description de modèles couramment employés.

1.1 Qu'est-ce que l'apprentissage automatique

L'apprentissage automatique est un sous-domaine de l'**intelligence artificielle** où sont conçus des algorithmes qui, via un apprentissage, sont en mesure d'extraire l'information caractérisant une distribution, puis d'accomplir certaines tâches liées à la distribution. L'algorithme aura comme entrée un ensemble de données que l'on appelle d'**apprentissage** ou d'entraînement, noté $\mathscr{D}$, et émettra en sortie un **modèle** ou une **fonction de décision** f. Une hypothèse souvent émise concernant les données est qu'elles sont indépendantes et identiquement distribuées (IID). Ce sera le cas pour tous les types de modèles mentionnés dans le présent mémoire. On parlera de **généralisation**, plutôt que de mémorisation, du moment où le modèle résultant sera en mesure d'obtenir une performance satisfaisante de prédiction lorsqu'appliqué à un nouvel ensemble de données, $\mathscr{D}_{\text{test}} = \{z_i; z_i \notin \mathscr{D}\}$. La composition des données dépend du type de tâche envisagée. Dans le cas d'une **tâche supervisée**, on voudra obtenir f qui permette d'associer les observations à leur cible, ou étiquette respective. Dans ce cas particulier, les données auront la forme suivante : $\mathscr{D} = \{z^{(i)} = (x^{(i)}, y^{(i)}); i = 1..n\}$ où $z^{(i)}$ représente une paire composée d'une entrée $x^{(i)} \in \mathbb{R}^d$ accompagnée de sa cible $y^{(i)}$. Cette dernière pourra être continue ou discrète, auquel cas l'on parlera de **régression** ou de **classification** respectivement. La tâche de classification implique des cibles retrouvées parmi un ensemble fini de classes, tandis que la tâche de régression fait correspondre un exemple à une statistique particulière de la distribution de la cible, conditionnelle à l'entrée. On voudra par exemple classifier des individus selon leur nationalité ou encore prédire, via la

régression, leur taille respective.

Quant aux tâches dites **non-supervisées**, il n'y a pas de cible associée aux entrées, ainsi $\mathscr{D} = \{z^{(i)} = x^{(i)}; i = 1..n\}$. Le but du modèle sera dès lors d'estimer au mieux la distribution des données dans le cas d'**estimation de densité**, ou certains aspects de cette distribution pour les autres types d'apprentissages non-supervisés. Certains algorithmes tels que l'algorithme des K moyennes [35] et les mixtures de gaussiennes permettent d'effectuer un **groupage** des données. L'apprentissage non-supervisé est aussi utilisé à des fin d'**extraction de caractéristiques**. Nous verrons plus loin que c'est le cas des empilements de machines de Boltzmann (RBM) et d'auto-encodeurs. Par ailleurs, l'apprentissage non-supervisé inclus aussi la **réduction de dimensionalité** où l'entrée x est transformée vers une représentation de plus petite dimension mais contenant l'essentiel de l'information. Cette dernière représentation pourra soit améliorer les performances d'une tâche supervisée, soit permettre une visualisation des données si la dimension résultante choisie est de 2 ou 3. À titre d'exemple, l'algorithme *Principal Component Analysis* (PCA) [37] préservera du mieux possible la variance des données de $\mathscr{D}$.

On dira qu'une tâche est **semi-supervisée**, dès lors qu'en plus des données étiquetées, sont utilisées des données supplémentaires sans étiquette. Ces dernières pourront servir au niveau d'une phase d'entraînement non-supervisé dans l'espoir d'améliorer la performance de l'algorithme d'apprentissage. Finalement, mentionnons l'**apprentissage** par renforcement comprenant la notion de récompense future et où un renforcement (positif ou négatif) est prodigué au modèle après l'accomplissement d'une action selon que cette dernière soit appropriée ou non considérant l'état du moment.

1.2 Les principes de l'apprentissage automatique

La phase d'apprentissage, consiste typiquement en la **minimisation d'un risque empirique**, soit la minimisation de la moyenne sur $\mathscr{D}$ d'une certaine perte ou coût $\mathscr{L}$. Considérant un modèle f définit par ses paramètres Θ, la minimisation du risque empirique consiste en la sélection des paramètres optimaux Θ^* qui viennent minimiser le

risque empirique $\mathscr{R}$ comme suit :

$$\Theta^* \leftarrow \underset{\Theta}{\operatorname{argmin}} \mathscr{R}(f, \mathscr{D}) = \underset{\Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(f_\Theta(x^{(i)}), y^{(i)}) \tag{1.1}$$

À noter que dans le cas non-supervisé, on utiliserait une fonction de perte $\mathscr{L}$ qui ne dépend pas de $y^{(i)}$. Par ailleurs, la notion de paramètre est associée à l'une des deux grandes familles des algorithmes d'apprentissage, soit le **modèle paramétrique** où un nombre fixe de paramètres Θ viennent caratériser la fonction f et sont responsables de lui donner une certaine capacité à la représentation des phénomènes observés dans $\mathscr{D}$. A l'opposé, les **modèles non-paramétrique** ont une capacité qui augmente avec le nombre d'exemple n d'entraînement et ne font pas de supposition quant à la famille de fonctions pouvant décrire la distribution de $\mathscr{D}$, certains vont plutôt utiliser l'information locale, située à l'intérieur d'une certaine distance du point de test, afin d'effectuer une prédiction. Ce qui cause d'ailleurs un problème en très haute dimension, car le nombre de données dans $\mathscr{D}$ nécessaires à couvrir une dimension d'entrée de $\mathbb{R}^d$ est exponentiel dans le nombre de dimensions. Ainsi, très rapidement, le nombre de données nécessaires à une prédiction de qualité devient inaccessible. C'est le **fléau de la dimensionnalité**.

Certains algorithmes admettent le contrôle explicite de la capacité du modèle en faisant varier la valeur de certains **hyper-paramètres**. Ces derniers sont des paramètres que l'on fixe avant l'exécution de l'entraînement. Un des hyper-paramètres typiques influençant la capacité du modèle est un coefficient de régularisation λ qui vient appliquer une contrainte sur les paramètres, diminuant la complexité du modèle. Nous reviendrons plus en détails sur la technique de régularisation lorsque nous aborderons la notion de généralisation dans la section 1.3.

L'une des méthodes utilisées pour la minimisation du risque empirique est la **descente de gradient**. Le gradient de $\mathscr{R}$ par rapport à chacun des paramètres donne la direction vers laquelle $\mathscr{R}$ augmente. Ainsi, en se déplaçant quelque peu dans le sens *contraire* du gradient, on permet au modèle de se rapprocher davantage d'un minimum de $\mathscr{R}$. Rien ne garantit toutefois que la minimisation sera complète ou encore que le minimum atteint sera un **minimum global** plutôt qu'un **minimum local**. Au choix, le

calcul du gradient peut être effectué sur un seul exemple à la fois (**descente stochastique**), sur l'ensemble $\mathscr{D}$ complet (**descente par *batch***) ou encore sur une certaine partie de ce dernier (**descente par *mini-batch***). Comme la descente stochastique est celle employée dans le cadre du présent mémoire, nous allons nous concentrer sur cette dernière. Il s'agit, pour chaque exemple de $\mathscr{D}$, de mettre à jour les paramètres en considérant le gradient de la perte calculée sur cet exemple, en fonction des paramètres actuels, soit $\frac{\partial \mathscr{R}}{\partial \Theta}$. Le pseudocode 1 montre cette procédure de descente de gradient stochastique.

Algorithm 1 Descente de gradient stochastique

```
Initialiser les paramètres θ
while La condition d'arrêt n'est pas atteinte do
  for all (x^(i), y^(i)) ∈ 𝒟 do
    for all θ_i ∈ Θ do
      Mettre à jour : θ_i ← θ_i − ε ∂ℛ/∂θ
    end for
  end for
end while
Retourner f_θ
```

1.3 La généralisation : le grand défi de l'apprentissage automatique

Ce qui fait la force d'un bon algorithme d'apprentissage est certainement sa capacité à la **généralisation**. C'est-à-dire qu'à l'opposé de la mémorisation, le modèle devra être en mesure de bien prédire sur des exemples jamais considérés jusqu'alors, ainsi, sa véritable performance sera celle estimée sur $\mathscr{D}_{\text{test}}$. La généralisation est conséquemment un défi important du domaine que l'on tentera de relever par divers moyens. Dès lors que l'algorithme nécessite des hyper-paramètres, on usera d'un troisième ensemble de données, soit $\mathscr{D}_{\text{valid}} = \{z_i; z_i \notin \{\mathscr{D}, \mathscr{D}_{\text{test}}\}\}$ qui servira à la **sélection du modèle**, soit la sélection de la combinaison d'hyper-paramètres offrant la plus basse erreur sur les exemples de $\mathscr{D}_{\text{valid}}$ qui va être un estimé biaisé de l'erreur de généralisation. Cette erreur s'avère effectivement biaisée, car il se pourrait que les hyper-paramètres choisis performent particulièrement bien sur $\mathscr{D}_{\text{valid}}$, on voudra donc tester en dernier lieu sur $\mathscr{D}_{\text{test}}$ pour l'obtention d'un estimé de l'erreur de généralisation cette fois-ci non biaisée. Un

risque encouru par les modèles paramétriques est le **sur-apprentissage** où la capacité du modèle est telle que l'optimisation sur $\mathscr{D}$ résulte en une prédiction quasi parfaite sur cet ensemble d'entraînement, mais médiocre sur $\mathscr{D}_{\text{test}}$, donc une très mauvaise généralisation. Une manière de contrer ce phénomène, est la diminution de la capacité du modèle, de manière à ce que ce dernier ne puisse pas s'ajuster trop *parfaitement* aux détails des données de $\mathscr{D}$ et soit en mesure de généraliser aux nouvelles données. Certains hyperparamètres permettent un tel ajustement de capacité dans les réseaux de neurones, par exemple le nombre d'unités par couche cachées, ou encore la **régularisation** appliquée aux paramètres. La régularisation se traduit par l'optimisation d'un risque empirique régularisé : plutôt d'optimiser 1.1 on va optimiser

$$\sum_{i=1}^{n} \mathscr{L}(f_\theta(x^{(i)}), y^{(i)}) + \lambda\,\omega(\theta) \tag{1.2}$$

où $\lambda\,\omega(\theta)$ induit une préférence pour certaines valeurs des paramètres. Différents types de régularisation existent tels que la régularisation *weights decay* de norme un (L1 : $\lambda\,\omega(\theta) = \sum_k |\theta_k|$) encourageant certains paramètres à être exactement nuls, ou de norme deux (L2 :$\lambda\,\omega(\theta) = \sum_k \theta_k^2$) forçant les poids à être près de zéro.

1.4 Modèles courants de classification

1.4.1 Modèle linéaire

Un modèle paramétrique très simple de classification à deux classes est le modèle linéaire. Il est défini par la fonction discriminante suivante :

$$g(x) = \mathbf{w}' \cdot x + b = \sum_j w_j x_j + b \tag{1.3}$$

soit **w** le vecteur de poids dans $\mathbb{R}^d$, déterminant la pente de la frontière de décision, et **b**, le biais dans $\mathbb{R}$, positionnant l'hyperplan séparateur ainsi défini dans l'espace. La fonction de décision associée, pour le cas des étiquettes $\in \{c_1, c_2\}$ sera : $f(x) =$

$\begin{cases} \mathbf{c}_2 \text{ si } g(x)>0 \\ \mathbf{c}_1 \text{ si } g(x)<0 \end{cases}$, c'est-à-dire que l'on prédira la classe du point de test selon qu'il se trouve d'une part ou de l'autre de l'hyperplan discriminateur. Un cas particulier d'algorithme apprenant un modèle linéaire, est l'algorithme du **Perceptron** [39] où le coût calculé sur un exemple est nul si aucune erreur de classification n'est observée, et proportionnel à $g(x)$ dans le cas contraire : $\mathscr{L}_{perceptron} = -g(x^{(i)})y^{(i)}I_{g(x^{(i)})y^{(i)}<0}$ avec $y^{(i)} \in \{-1,1\}$ et $I_u = \begin{cases} 1 \text{ si } u \text{ est vrai} \\ 0 \text{ sinon} \end{cases}$. Ainsi, plus l'exemple mal classifié sera loin de l'hyperplan, plus la pénalité associée en valeur absolue sera élevée. Une descente de gradient peut dès lors être effectuée sur ce coût afin de trouver les paramètres optimaux $\Theta^* = (\mathbf{w}, b)$.

Une autre variante d'un modèle de classification linéaire, est le **régresseur logistique**, dont la sortie peut se traduire en la probabilité d'appartenance à la classe 1, soit $g(x) \cong p(y=1|x) = 1-p(y=0|x)$. Pour se faire, la fonction discriminante (équation 1.4) résulte d'une non-linéarité de type sigmoïde (équation 1.4) appliquée sur la transformation linéaire de l'entrée, tel que sigmoïde $:[-\infty,+\infty] \rightarrow [0,1]$. La fonction de coût associée (équation 1.6) est l'entropie croisée admettant une pénalité proportionnelle au logarithme négatif de la probabilité prédite que $x^{(i)}$ appartienne effectivement à la classe $y^{(i)}$. Quant à la fonction de décision, il suffit de trancher avec le seuil approprié (équation 1.5).

$$g(x) = \text{sigmoide}(w'x+b) = \frac{1}{1+\exp^{-(w'x+b)}} \tag{1.4}$$

$$f(x) = \begin{cases} 1 \quad (\text{class } \mathbf{c}_2) & \text{si g(x) >= 0.5} \\ 0 \quad (\text{class } \mathbf{c}_1) & \text{si g(x) < 0.5} \end{cases} \tag{1.5}$$

$$\mathscr{L}_{Reg.Log.} = -y^{(i)}\log(g(x^{(i)})) - (1-y^{(i)})\log(1-g(x^{(i)})) \tag{1.6}$$

Un autre algorithme développé au départ pour apprendre un modèle linéaire est celui des **machines à vecteur à support (SVM)**. Il s'agit d'un algorithme apprenant une fonction discriminante linéaire de la forme $g(x) = \mathbf{w}' \cdot x + b$, avec la particularité d'obtenir une marge maximale, soit une distance maximale entre l'hyperplan et l'exemple d'entraînement le plus proche. Ce modèle du SVM linéaire, et les précédents sont ap-

propriés pour des données où les deux classes sont linéairement séparables. Afin d'être en mesure de bien prédire sur des données non-linéairement séparables et ce, à l'aide d'un modèle linéaire, il est possible d'appliquer un pré-traitement non-linéaire φ sur les données x, tel que

$$\varphi : x \in R^d \rightarrow \widetilde{x} \in R^k \text{où } k > d, \tag{1.7}$$

de manière à ce que les données ainsi transformées en dimension k soient (quasi) linéairement séparables. De là, trois méthodes sont envisageables : choisir à priori la fonction de transformation φ et appliquer explicitement le pré-traitement afin d'obtenir $\widetilde{x} = \varphi(x)$, déterminer implicitement la transformation par le biais de **l'astuce du noyau** [1] qui se base sur le calcul du produit scalaire $\left\langle x^{(i)}, y^{(j)} \right\rangle$, sans jamais avoir à calculer explicitement la transformation $\varphi(x)$, finalement, il est possible d'*apprendre* la transformation non-linéaire. Une transformation explicite fixe telle une transformation polynomiale d'ordre k, comporte un problème : du moment que x est de haute dimension, une telle transformation implique de calculer un $\widetilde{x}$ de très haute dimension. L'astuce du noyau quant à elle peut être appliquée à n'importe quel algorithme qui peut se traduire par le biais de produit scalaire. On compte parmi eux la régression logistique, le perceptron et les SVMs. Le SVM avec, par exemple, un noyau gaussien (SVM_{rbf}) ou un noyau polynomial (SVM_{poly}), pouvant dès lors produire un classifier non linéaire devient très prisé, et sera souvent utilisé à titre de référence, étant donné sa bonne performance et son petit nombre d'hyper-paramètres. Finalement, l'apprentissage de la transformation peut être effectué grâce à la première couche cachée d'un réseau artificiel de neurone. À la prochaine section, nous verrons plus en détail le réseau de neurones artificiel qui est le modèle non-linéaire de base à partir de quoi découleront les expérimentations issues de la présente étude.

1.4.2 Réseau de neurones artificiel

Le réseau de neurones artificiel (ANN), ou **perceptron multi-couche (MLP)**, peut être perçu comme une régression logistique appliquée à une transformation non-linéaire

apprise de l'entrée. Tel que précisé précédemment, cette représentation offrira au modèle une capacité supérieure, lui permettant d'effectuer des tâches de classification et de régression *non-linéaire*. Le réseau de neurones est constitué d'une couche d'entrée de dimension d, recevant un vecteur d'observations, d'un certain nombre de couches cachées composées d'unités cachées appliquant chacune une certaine non-linéarité à une projection linéaire de leur entrée, et finalement, une couche de sortie de dimension m déterminée en fonction de la tâche à effectuer : classification binaire versus multi-classe, régression simple versus multiple. Si l'on prend le cas d'un réseau de neurones à une seule couche cachée comportant d_h unités cachées, il calcule en sortie la fonction $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$, telle que :

$$f(x) = g(c + V(h(b + Wx))), \tag{1.8}$$

avec les vecteurs de biais $b \in R^{d_h}$ et $c \in R^m$, les matrices de poids $W \in R^{d_h \times d}$ et $V \in R^{m \times d_h}$ et finalement les fonctions d'activation G et H telles que la fonction sigmoïde et la fonction tangente hyperbolique :

$$\text{sigmoide}(a) = \frac{1}{1+\mathbf{e}^{-a}} : [-\infty, +\infty] \rightarrow [0,1] \tag{1.9}$$

$$\tanh(a) = \frac{\mathbf{e}^{a} - \mathbf{e}^{-a}}{\mathbf{e}^{a} + \mathbf{e}^{-a}} : [-\infty, +\infty] \rightarrow [-1,1] \tag{1.10}$$

À noter que si a est un vecteur, les fonction tanh et sigmoide s'appliquent indépendamment à chacun de leurs éléments pour donner en sortie un vecteur de la même dimension. Ces fonctions d'activations à la couche cachée permettent l'intégration d'une non-linéarité dans la prédiction, tandis que celles à la couche de sortie permettent l'obtention d'une nature et d'un intervalle de valeurs compatibles avec la tâche à accomplir. On pourra par exemple utiliser la sigmoïde déjà mentionnée pour une classification binaire, ou encore la fonction softmax :

$$softmax(a)_i = \frac{e^{a_i}}{\sum_j e^{a_j}} \tag{1.11}$$

qui produit un vecteur de même dimension que a tel que chacun de ses éléments a une valeur $\in [0,1]$ et somment à 1. Cette dernière non-linéarité est fréquemment utilisée dans le cas de la classification multi-classe car elle permet l'obtention en sortie des probabilités d'appartenance à chacune des classes.

Les différentes étapes nécessaires à l'apprentissage des paramètres $\theta = [W,b,V,c]$ sont les suivantes : calculer le gradient de la fonction de perte L sur un exemple par rapport à la sortie du réseau. Ceci se fait efficacement avec la technique de **rétro-propagation du gradient** [32, 40], qui est une application efficace de la règle de dérivation en chaîne, jusqu'au bas de l'architecture. On obtient ainsi au final le gradient de la perte par rapport à chacun des paramètres. Finalement, on met à jour ces derniers à l'aide de la règle d'apprentissage, soit par exemple pour le cas des poids W : $W \rightarrow W - \eta \frac{\partial L}{\partial W}$, où η est un pas d'apprentissage, et $\frac{\partial L}{\partial W}$ le gradient de la perte L par rapport aux paramètres W.

Selon que l'on utilise la descente de gradient stochastique, par mini-batch ou par batch, la mise à jour des paramètres se fera soit en utilisant le gradient sur un exemple, une batch ou leur somme cumulée sur tous les exemples d'entraînement, respectivement.

Les différents niveaux de non-linéarité du MLP, résultant en une capacité supérieure, font naître deux problèmes d'envergure. En effet, il s'en suit que la fonction résultante à optimiser est non convexe, impliquant qu'il soit fort probable que le minimum atteint soit en fait un minimum local, plutôt que global. La descente stochastique, de par sa nature stochastique, permet parfois d'échapper à certains minima locaux [10, 31]. De plus, une forte capacité est accompagnée du risque accru de sur-apprentissage. Pour contrer ce dernier phénomène, on pourra diminuer la capacité du modèle en diminuant son nombre d'unités cachées, ou encore lui imposer une régularisation.

Le perceptron multi-couche à une couche cachée a la propriété d'être un **approximateur universel** [24], c'est-à-dire qu'avec un nombre suffisant d'unités cachées, il est

en mesure d'apprendre à représenter n'importe quelle fonction continue. Nous verrons toutefois au chapitre 2 qu'il est parfois coûteux d'accomplir convenablement ce rôle (en un sens qui sera sera précisé dans le chapitre suivant). Il s'agit là de l'une des raisons motivant l'utilisation de réseaux profonds.

CHAPITRE 2

RÉSEAU DE NEURONES PROFOND ET MÉTHODE D'ENTRAÎNEMENT

Les réseaux de neurones peu profonds possèdent des limites qui ont poussé des chercheurs à faire l'emploi de réseaux de neurones possédant davantage de couches cachées, architecture que l'on dit profonde. Nous verrons dans ce chapitre les limites des réseaux peu profonds et les bénéfices apportés par l'utilisation d'architectures profondes. Nous discuterons des principaux défis associés à une telle architecture, dont un en particulier qui aura grandement nuit à sa popularité : sa difficulté d'entraînement. Nous introduirons la méthode qui a récemment permis de surpasser cette difficulté, soit un pré-entraînement initialisant les paramètres du réseau, couche après couche, selon un critère non-supervisé. Cette procédure de pré-entraînement servira de base au présent mémoire. Pour un survol traitant des réseaux de neurones profonds, voir [6].

2.1 Motivations attribuées à l'usage d'une architecture profonde

Il a été démontré qu'un réseau à une seule couche cachée de largeur illimitée pouvait représenter n'importe quelle fonction [24]. Toutefois, comme l'aura démontré [7, 18] certaines familles de fonctions pouvant être représentées par un réseau de neurones de profondeur k, nécessitent un nombre exponentiel d'unités afin d'être représentées par une architecture de profondeur moindre, soit de $k-1$. Ainsi, afin qu'un réseau peu profond puisse bien représenter n'importe quelle fonction, il devra posséder un nombre exponentiel de paramètres. On sait par ailleurs qu'une bonne généralisation passe par un nombre d'exemples d'entraînement qui croît au moins linéairement avec le nombre de paramètres dans le réseau [4]. Il s'ensuit qu'une architecture peu profonde est inefficace en terme de paramètres nécessaires et donc de nombre nécessaire d'exemples [7, 8].

De plus, des études théoriques ont démontré que les architectures profondes étaient indispensables à une modélisation efficace de distributions complexes pour l'obtention d'un meilleur niveau de généralisation [6, 7]. De par leurs différentes couches non-

linéaires, ces réseaux sont en mesure de mieux représenter des fonctions non-linéaires et hautement variées que les réseaux non profonds.

Par ailleurs, la notion de hiérarchie est présente dans plusieurs sphères qui composent notre monde. La musique par exemple possède comme unité de base la note, qui jointe avec d'autres notes composent une mesure, et ainsi de suite, jusqu'à l'obtention d'une partition musicale. Un second exemple est le langage, formé de lettres, de mots puis de phrases. Aussi, des travaux qui tentent de mieux comprendre le fonctionnement des différentes sphères du cerveau ont mis en évidence le caractère hiérarchique de ces dernières. Notamment, l'aire visuelle, composée des régions V1 à V5, intègre une certaine notion de hiérarchie à l'intérieur de son fonctionnement. Les stimuli en provenance de l'oeil, circulent vers la région V1 puis vers les subséquentes, chacune des régions se chargeant d'extraire une particularité des stimuli d'entrée. Ainsi, V1 est notamment responsable de l'extraction des contours de l'image [25], tandis que V2 détecte des patrons plus complexes, tels qu'un regroupement de contours [19, 26]. En lien avec ces recherches expérimentales, des travaux d'apprentissage automatique ont démontré qu'un entraînement non-supervisé sur des données d'images naturelles pouvait admettre l'apprentissage de patrons semblables à ceux retrouvés au sein des cellules V1 [36] et V2 [33] du cortex visuel. Conséquemment, un argument supplémentaire en faveur de l'exploration des réseaux de neurones profonds, est la possibilité de décortiquer l'information d'entrée et d'apprendre des représentations de plus en plus abstraites, à mesure que l'on s'élève dans l'architecture.

Pour toutes ces raisons, des chercheurs se sont penchés sur l'étude des architectures profondes. Toutefois, en pratique, ils ont dû faire face à une difficulté de taille, à savoir l'entraînement efficace d'un tel réseau à plusieurs couches.

2.2 Défis de l'entraînement des architectures profondes

La méthode générale d'entraînement d'un réseau de neurones, profond ou non, consiste en la transformation d'une tâche supervisée donnée en un problème d'optimisation, où la fonction à minimiser représente un coût supervisé influencé par la prédiction du ré-

seau et la cible que l'on désire approcher. On emploie la méthode d'optimisation qu'est la descente de gradients afin d'ajuster peu à peu les paramètres du système jusqu'à ce que le coût sur les exemples d'entraînement devienne minimal ou qu'un critère d'arrêt soit atteint. Malheureusement, un tel entraînement supervisé basé sur une initialisation aléatoire des paramètres d'un réseau de plus d'une ou deux couches cachées se solde habituellement par de mauvaises performances de généralisation [7, 14, 22, 30].

Une hypothèse permettant d'expliquer ce pauvre résultat serait l'apparition supplémentaire d'un nombre considérable de mauvais minima locaux avec l'augmentation du nombre de couches cachées de l'architecture. En effet, le critère d'entraînement étant non convexe, on est confronté à de bons comme à de mauvais minima locaux et plateaux, et rien ne nous garantit que la descente de gradient ne restera pas coincée à l'intérieur de l'un de ces mauvais minima ou plateaux locaux [3]. Par ailleurs, le nombre de mauvais minima influence clairement les chances que l'initialisation des paramètres se situe dans le bassin d'attraction de l'un d'eux [16]. Conséquemment, si l'augmentation du nombre de couches cachées vient décupler le nombre de mauvais minima locaux, il s'ensuit une très forte probabilité d'aboutir dans l'un d'eux, résultant en une solution pauvre même au niveau des données d'entraînement. Afin de contrer cette problématique, il a été suggéré de sous-diviser la difficulté du problème d'optimisation en plusieurs étapes voraces et simples. En guise d'exemple, [15] a proposé une méthode d'entraînement faisant l'ajout d'un neurone à la fois, tandis que [34] ont généralisé à l'ajout d'une couche à la fois.

La seconde façon d'expliquer une mauvaise généralisation issue de l'entraînement d'un réseau de neurones profond est la suivante. Un réseau de neurones profond possédant une grande capacité et flexibilité, se voit posséder dans l'espace de ses paramètres plusieurs bassins d'attraction qui, selon l'initialisation, seront ou non visités. Ainsi, le réseau obtiendra une solution différente en fonction de l'initialisation de ses paramètres. Certaines solutions pourraient s'avérer provenir d'un bon minimum local en terme de l'ensemble d'entraînement, mais rien ne garantit qu'elles seront par la même occasion de bonnes solutions en terme d'exemples de test. En d'autres mots, une bonne prédiction sur l'ensemble d'entraînement n'assure pas une généralisation adéquate, et l'on parlera de sur-apprentissage si la performance de généralisation est mauvaise. Entre en jeu la

sélection du modèle dont la capacité à la généralisation est la plus adéquate. Pour ce faire, il faut d'abord que la procédure d'entraînement soit en mesure de visiter les bons minima locaux en terme de généralisation, ce qui n'est pas gagné d'avance si la proportion de configurations propres à une bonne généralisation est faible par rapport à celles résultant en un sur-apprentissage.

Nous verrons dans la section suivante, une technique dite de pré-entraînement non-supervisé qui permet l'atteinte de meilleurs bassins d'attraction [14] pour l'obtention d'une bonne performance de généralisation à la tâche supervisée.

2.3 Pre-entraînement non supervisé

Une manière bien connue de diminuer le sur-apprentissage et améliorer la généralisation est l'usage de la régularisation. Tel que spécifié précédemment, la régularisation permet de privilégier un modèle plus simple, de capacité moindre, de manière à éviter une configuration des paramètres qui s'avère idéale en terme d'entraînement, mais mauvaise en terme de généralisation. Toutefois, l'application d'une contrainte à priori telle qu'imposée par la régularisation L2, ne transmet que très peu d'information sur ce que devrait être la solution. C'est ce qui a motivé plusieurs chercheurs à développer de nouvelles stratégies de régularisation, cette fois-ci dépendantes de la distribution d'entrée et basées sur un apprentissage non-supervisé.

Ainsi, en 2006, une percée a eu lieu concernant l'entraînement d'un réseau de neurones profond : [22] a suggéré une approche efficace qui vient remplacer l'initialisation aléatoire des poids et biais d'un réseau. La technique combine l'idée de la division du problème d'optimisation en sous-étapes plus faciles, avec celle d'utiliser un entraînement non-supervisé afin de donner un bon indice aux unités cachées de ce qu'elles devraient apprendre. Par certains aspects, cette approche agit comme régularisateur [14], de sorte que malgré la haute capacité du réseau de neurones, l'algorithme évite le sur-apprentissage. La technique est appelée **pré-entraînement non-supervisé** et consiste en l'utilisation d'un critère non-supervisé pour l'initialisation des poids et des biais, couche après couche, avant d'entamer l'entraînement supervisé du réseau, tel qu'on le

connait. C'est durant cette phase d'apprentissage non-supervisé que la première couche cachée tente d'apprendre à modéliser la distribution de l'entrée du réseau, puis alimente la couche suivante, qui à son tour, apprendra à modéliser la distribution de la couche précédente et ainsi de suite, jusqu'à ce que la dernière couche cachée ait été entraînée (pseudo-code 2). En d'autres mots, le réseau apprend, au niveau de chacune de ses couches cachées, une représentation de la couche précédente. Cette représentation est dite **sous-complète**, **complète** ou **sur-complète**, selon qu'elle est de taille inférieure, égale ou supérieure à la couche précédente, respectivement.

Algorithm 2 Pseudo-Code du pré-entraînement non-supervisé

```
initialiser les poids W et biais b de toutes les couches, y compris la couche de sortie
for all Couche cachée k ∈ {1,..,l} do
  while la condition d'arrêt n'est pas atteinte, itérer sur les exemples x^(i) de 𝒟, do
    obtenir les représentation h^(k−1)(x^(i)) (si i>1) et h^k(x^(i))
    en utilisant le critère non-supervisé, optimiser les poids W_k et biais b_k
  end while
end for
```

L'inventeur de cette technique [22] l'a appliquée avec succès sur une architecture d'empilement de machines de Boltzmann restreintes, soit un *Deep Belief Network* (DBN) composé de *Restricted Boltzmann Machines* (RBM). Depuis, l'idée a fait son chemin et différentes alternatives à l'empilement de RBM ont été étudiées tels l'empilement d'auto-encodeurs ordinaires (SAE) [5, 38], l'empilement d'auto-encodeurs débruiteurs [44] (SDAE), l'utilisation de représentations dites semi-supervisées [47] et noyau PCA [13]. Dans le cadre du présent mémoire, nous nous attarderons aux architectures DBN, SAE et plus particulièrement SDAE dont les modules respectifs servant à leur pré-entraînement sont décrits aux sections suivantes.

2.3.1 Machines de Boltzmann restreintes (RBM)

Les **machines de Boltzmann restreintes (RBMs)** [43] sont des modèles génératifs basés sur une fonction d'énergie qui dépend d'un vecteur de composantes d'entrée (x) et d'un vecteur de composantes cachées binaires et stochastiques (h). Le modèle tentera de capter la structure de l'entrée de manière à pouvoir modéliser la distribution de cette

dernière. Bien que modélisant habituellement des distributions binaires, le modèle a été étendu à des unités d'entrées non binaires [22, 46]. À la différence d'une machine de Boltzmann [20], la machine de Boltzman *restreinte* [43] ne possède pas de connexion entre ses unités d'une même couche. La probabilité jointe de la couche d'entrée avec celle cachée est la suivante :

$$p(x,h) = \frac{\mathbf{e}^{-\mathbb{E}(x,h)}}{Z} \tag{2.1}$$

où Z assure que $p(x,h) \in [0,1]$ et somme à 1, tandis que $\mathbb{E}(x,h)$ est la fonction d'énergie définie comme suit :

$$\mathbb{E}(x,h) = -\sum_{jk} W_{jk} x_k h_j - \sum_k c_k x_k - \sum_j b_j h_j, \tag{2.2}$$

avec le vecteur d'entrée $x \in R^d$, le vecteur caché $h \in R^{d_h}$. L'ensemble des paramètres du modèle est $\theta = \{W, b, c\}$. Il est composé de : la matrice de connexion entre la couche d'entrée et la couche cachée $W \in R^{d_h \times d}$, les biais ajoutés à l'activation de la couche cachée $b \in R^{d_h}$ et les biais ajoutés à l'activation de la couche d'entrée $c \in R^d$.

Dans ces conditions, on a :

$$p(x|h) = \prod_k p(x_k|h) \quad \text{où} \quad p(x_k = 1|h) = \text{sigmoide}(\sum_j W_{jk} h_j + c_k) \tag{2.3}$$

$$p(h|x) = \prod_j p(h_j|x) \quad \text{où} \quad p(h_j = 1|x) = \text{sigmoide}(\sum_k W_{jk} x_k + b_j) \tag{2.4}$$

Lors de l'entraînement d'une RBM, on cherche idéalement les paramètres θ qui maximisent la log vraisemblance moyenne sur l'ensemble de données d'entraînement $\mathscr{D}$:

$$\frac{1}{N} \sum_{i=1}^{n} \log p(x = x^{(i)})$$

La contribution d'un exemple $x^{(i)}$ au gradient de cette log vraisemblance moyenne par rapport aux paramètres θ peut s'exprimer ainsi :

$$\left.\frac{\partial \log p(x)}{\partial \theta}\right|_{x=x^{(i)}} = -\sum_h p(h|x=x^{(i)}) \frac{\partial \mathbb{E}(x^{(i)},h)}{\partial \theta} + \sum_{x,h} p(x,h) \frac{\partial \mathbb{E}(x,h)}{\partial \theta} \tag{2.5}$$

Ce calcul est ardu et impossible à réaliser en pratique. C'est pourquoi les RBMs sont typiquement entraînées avec une technique donnant une estimation approximative et biaisée de ce gradient, la **Divergence Contrastive** [12]. Nous nous contentons ici de donner les détails de l'algorithme pratique qui l'implémente, et référons le lecteur à [12] pour la motivation de cette approximation et son exposition davantage formelle. Traduite en un algorithme détaillé, la technique se décompose de trois étapes :

La première étape, dite **phase positive** consiste en le calcul des unités de couche cachée comme suit :

$$x^0 \leftarrow x \tag{2.6}$$

$$\hat{h}^0 \leftarrow \text{sigmoide}(b + W x^0) \tag{2.7}$$

La **phase négative**, quant à elle, consiste en les opérations suivantes ;

$$h^0 \sim p(h|x^0) \tag{2.8}$$

$$x^1 \sim p(x|h^0) \tag{2.9}$$

$$\hat{h}^1 \leftarrow \text{sigmoide}(b + W x^1) \tag{2.10}$$

Finalement, la troisième étape consiste en la **mise à jour des paramètres** à l'aide des gradients calculés :

$$W^i \leftarrow W^i + \eta * (\hat{h}^0(x^0)^T - \hat{h}^1(x^1)^T) \tag{2.11}$$

$$b^i \leftarrow b^i + \eta * (\hat{h}^0 - \hat{h}^1) \tag{2.12}$$

$$c^i \leftarrow c^i + \eta * (x^0 - x^1) \tag{2.13}$$

2.3.2 Auto-encodeur ordinaire (AE)

Suite à l'importante découverte de [22] utilisant des RBM pour le pré-entraînement, [5] a appliqué la technique sur l'empilement d'un module quelque peu différent, soit un auto-encodeur ordinaire (AE pour *Auto-Encoder*). Le principe de l'AE se divise en deux phases. La première, l'*encodage*, consiste en une transformation déterministe de l'entrée, un vecteur $x \in [0,1]^d$, d'abord linéaire puis suivie d'une non-linéarité donnant lieu à une représentation cachée de l'entrée : $y = g_\theta(x) = H(Wx+b)$ où H est une non-linéarité au choix, $y \in [0,1]^{d_h}$, puis θ est défini par W, une matrice $\in R^{d_h \times d}$, et b, un vecteur de biais $\in R^{d_h}$. La seconde phase est celle du *décodage*, où l'AE obtient le vecteur de sortie z via une transformation linéaire suivie optionnellement d'une transformation non-linéaire, c'est-à-dire : $f_{\theta'}(y) = Vy + c$ ou encore $f_{\theta'}(y) = H(Vy+c)$ avec H, une non-linéarité, puis θ' formé de V, soit la matrice transposée de W s'il s'agit de **matrices liées**, et un biais c de dimension d. Ces étapes d'encodage et de décodage sont illustrées à la figure 2.1(a). Un bon AE sera en mesure d'encoder puis de décoder de manière à retrouver en sortie un vecteur z s'apparentant significativement à l'entrée x. Ainsi, l'entraînement consiste en l'apprentissage des paramètres θ et θ' qui minimisent la moyenne, sur l'ensemble d'apprentissage, d'une erreur de reconstruction L (fonction de x et z) :

$$\theta^*, \theta'^* = argmin_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, z^{(i)}) \tag{2.14}$$

$$\tag{2.15}$$

où n est le nombre d'exemples d'apprentissage. L'application ou non d'une non-

linéarité au niveau de z, de même que le choix de L dépend du type de données d'entrées. On fait usage d'une non-linéarité sigmoïde et préférablement de l'entropie croisée pour des données d'entrée binaires ou quasi binaires. L'entropie croisée pour un exemple est définie comme suit :

$$L(x,z) = -\sum_{k=1}^{d} [x_k \log(z_k) + (1-x_k)\log(1-z_k)]. \tag{2.16}$$

Tandis qu'on n'utilise aucune non-linéarité à la sortie du module et fait usage du coût quadratique lorsque les composantes sont de valeurs continues qui n'ont rien de binaire. La perte résultante pour un exemple est la suivante :

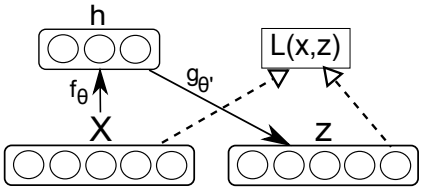
$$L(x,z) = \sum_{k=1}^{d} [(x_k - z_k)^2] \tag{2.17}$$

Les paramètres sont optimisés à l'aide d'une descente de gradient, qui sera, tout au long de nos travaux, stochastique.

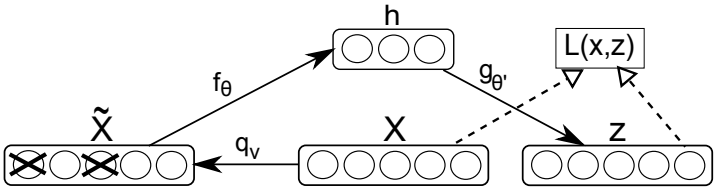
2.3.3 Auto-encodeur débruiteur (DAE)

Considérant que la qualité des représentations de plus haut niveau de l'entrée pourrait être la clef d'une meilleure généralisation, [45] se sont penchés sur la question à savoir qu'est-ce qui pourrait augmenter la qualité d'une telle représentation apprise. De cette réflexion est né un critère additionnel permettant une représentation robuste à une entrée partiellement détruite. L'idée étant de forcer l'architecture à capter des structures et régularités particulièrement intéressantes de la distribution, et suffisantes à la reconstruction de l'entrée malgré sa destruction partielle. Autrement dit, il s'agit d'apprendre une représentation utile au *débruitage* de l'entrée.

L'auto-encodeur débruiteur (ou DAE pour *Denoising Auto-Encoder*), ainsi proposé ne diffère de l'auto-encodeur ordinaire (AE) que par une étape supplémentaire, soit la corruption (ou "bruitage") de l'entrée x, devenant $\widetilde{x}$, avant de la transformer en une re-



(a) Auto-encodeur ordinaire (AE)



(b) Auto-encodeur débruiteur (DAE)

Figure 2.1 – **Entraînement d'un auto-encodeur ordinaire (AE) et d'un auto-encodeur débruiteur (DAE).** Dans les deux cas, l'entraînement consiste à optimiser, jusqu'à l'atteinte d'un critère d'arrêt, les paramètres θ et θ' de manière à minimiser la moyenne sur l'ensemble d'apprentissage de "l'erreur de reconstruction" $L(x,z)$ entre x et sa reconstruction z. Pour le cas plus complexe qu'est celui du DAE, les étapes sont les suivantes : un exemple x est partiellement corrompu via la transformation q_ν selon une certaine fraction ν de composantes corrompues, pour devenir $\widetilde{x}$. Ensuite, la fonction $f_\theta(\widetilde{x})$ transforme non-linéairement l'entrée corrompue $\widetilde{x}$ résultant en une représentation "cachée" h, et finalement, $z = g_{\theta'}(h)$ est le vecteur reconstruit que l'on voudra le plus semblable possible à l'entrée non bruitée x.

présentation cachée $y = f_\theta(\widetilde{x})$, puis d'effectuer la reconstruction $z = g_{\theta'}(f_\theta(\widetilde{x}))$. Le type de destruction de l'entrée qui a été expérimenté par [45] est l'application d'un *masque à zéro* (ou *masking noise* (MN)) sur une fraction ν des composantes d'entrée. C'est-à-dire qu'à chaque nouvel exemple, une fraction des composantes de l'exemple est choisie au hasard et leur valeur est remplacée par 0. Sur les ensembles de données utilisés, ce bruit peut être interprété comme de petites perforations de l'image que le DAE devra apprendre à combler, notamment en *reconstruisant* ou en *débruitant* x. Au cours de nos recherches, nous serons amenés à expérimenter de nouveaux types de bruit qui seront introduits dans le chapitre 4.

La perte dite *de reconstruction* est calculée entre le vecteur reconstruit z et l'exemple original non corrompu x. De la même manière que pour le AE, l'application ou non d'une non-linéarité au niveau de la couche de reconstruction, ainsi que le choix du coût

de reconstruction dépendront de la nature des composantes d'entrées du DAE. Les paramètres θ et θ' sont optimisés de la même façon que dans le cas du AE, par descente de gradient stochastique. L'image 2.1(b) schématise ces différentes étapes, avec en comparaison celles de l'AE. Un bon DAE sera en mesure de *reconstruire* du mieux possible l'entrée qui a été corrompue. Aussi, contrairement au AE, comme le vecteur qui est encodé, soit $\widetilde{x}$, est différent du vecteur x que le DAE tente de reconstruire, nous sommes assurés que le DAE ne peut se contenter d'apprendre une solution triviale (telle que $g(f(x)) = x$). En ce sens, cela nous évite de devoir utiliser une autre forme de régularisation, ou encore de se restreindre à faire l'apprentissage de représentations *sous-complète*, afin d'empêcher l'atteinte d'une telle solution triviale.

L'idée d'entraîner à l'aide d'une tâche de débruitage n'est pas nouvelle. Elle a été introduite par [17, 32] pour l'apprentissage d'une mémoire auto-associative via des composantes d'entrées binaires dont une certaine fraction choisie aléatoirement est inversée. Par après, [41] a entraîné un réseau de neurones récurrent pour une tâche de complétion de motif. Finalement [27], inspiré par [5, 41] a présenté une seconde approche pour le débruitage d'images. Bien que les modèles et la procédure d'entraînement soient semblables au cas du DAE, la motivation en est tout autre : alors que [45] désire améliorer le pré-entraînement menant à une initialisation plus adéquate des couches d'un réseau profond pour la complétion d'une tâche supervisée, [17, 32] se concentraient sur la capacité du modèle lors d'une tâche de mémorisation, tandis que [41] mettait son emphase sur le réseau de neurones récurrent et, tout comme [27], sur la tâche de débruitage en tant que telle d'une image.

2.4 Entraînement supervisé

L'initialisation des poids d'une architecture profonde peut être réalisée grâce à l'empilement et l'entraînement successif de modules de type RBM, AE ou DAE. On nomme ces architectures respectives, un DBN *(Deep Belief Net)*, un SAE *(Stacked Auto-Encoder)*, et un SDAE *(Stacked Denoising Auto-Encoder)*. Après avoir entraîné le k^e module convenablement selon son critère non-supervisé, on poursuit avec l'entraînement du $(k+1)^e$

module, utilisant la représentation obtenue au niveau de la couche cachée du k^e module en guise d'entrée pour le $(k+1)^e$. Il est important d'observer que dans le cas où le vecteur d'origine subit une corruption comme pour le cas du module DAE, cette version corrompue n'est plus utilisée dès lors que l'entraînement du module est terminé. Pour référence, l'image 2.2 montre les étapes de pré-entraînement d'une architecture utilisant l'empilement de deux AE, tandis que l'image 2.3 montre ces mêmes étapes pour le cas d'une architecture à RBM empilées. Au final, les poids appris de chacun des modules serviront de poids initiaux pour la prochaine étape, soit l'entraînement selon un critère supervisé.

Par-dessus cet empilement de modules vient s'ajouter une couche de sortie à m neurones appliquant une certaine non-linéarité selon la nature désirée des sorties. Nos travaux étant basés sur des problèmes de classification, nous utilisons un nombre de neurones de sortie égal au nombre de classes, et appliquons une non-linéarité de type *softmax* qui permet l'obtention de valeurs en sortie $\in [0,1]$ sommant à 1. Il nous est donc possible d'interpréter la valeur calculée par chacun des neurones de sorties comme la probabilité que l'exemple en cours fasse partie la classe associée. Le coût que l'on voudra dès lors minimiser, consiste en l'opposé du logarithme de la sortie associée à la cible réelle, soit : $-\log(s_y)$ où s est le vecteur de sortie et y ; la cible réelle $\in [0, m-1]$. La phase d'apprentissage supervisée consiste à faire l'usage de la descente de gradient stochastique afin d'optimiser les paramètres associés à chacune des couches de sorte que l'erreur em-

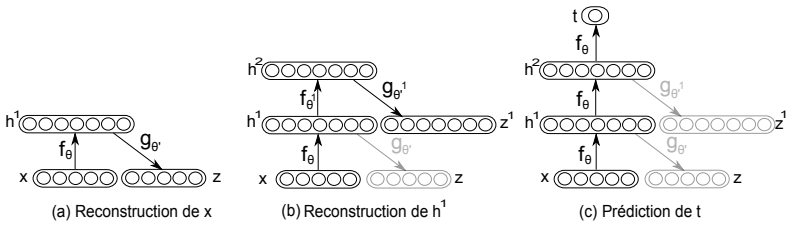


Figure 2.2 – **Pré-entraînement couche après couche d'un SAE.** Entraînement de la première (a) puis seconde (b) couche du réseau. On trouve les paramètres θ et θ^1 en optimisant un critère d'apprentissage non-supervisé propre au DA. Une fois le pré-entraînement complété, l'architecture est prête à être entraînée selon un critère supervisé, qui pour chaque exemple vise à l'obtention d'une sortie t prédictive de la cible supervisée.

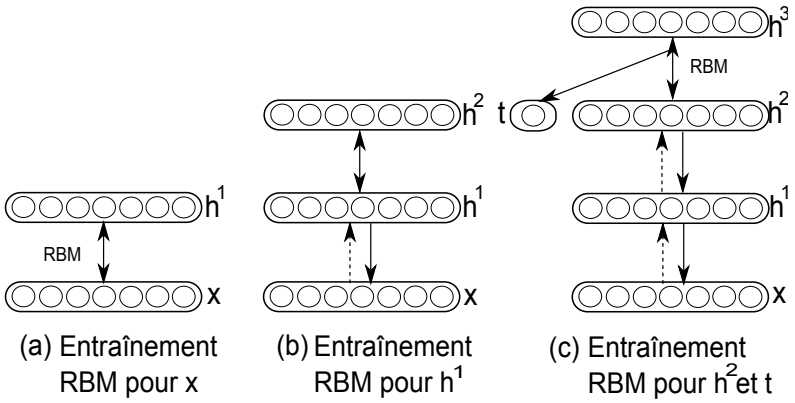


Figure 2.3 – **Pré-entraînement couche après couche d'un DBN.** Entraînement de la première (a) puis seconde (b) couche du réseau. On trouve les paramètres θ et θ^1 en entraînant chaque RBM avec la technique de Divergence Contrastive. Une fois le pré-entraînement complété, l'architecture est prête à être entraînée selon un critère supervisé, qui pour chaque exemple vise à l'obtention d'une sortie t prédictive de la cible supervisée.

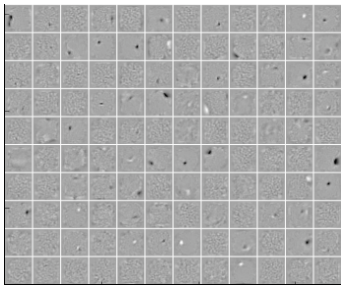
pirique soit minimisée, et ce, jusqu'à un maximum de passage sur $\mathscr{D}$ ou, si l'on utilise **l'arrêt prématuré**, jusqu'à ce que l'erreur sur $\mathscr{D}_{\text{valid}}$ ne diminue plus ou encore qu'elle se mette à augmenter, signe d'un sur-apprentissage.

2.5 Validation du pré-entraînement

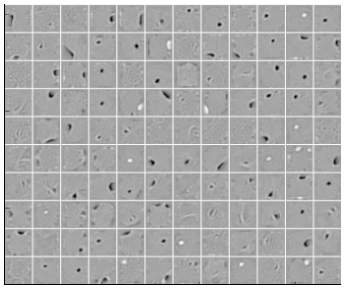
Suite au succès en terme de performance de classification du pré-entraînement d'un réseau profond à l'aide d'un empilement de RBM [22], il a été montré que le remplacement des RBM par des AE permettait l'obtention d'une performance de généralisation presqu'aussi bonne [5, 30]. L'étude empirique de [14] montre que ces techniques de pré-entraînement permettent l'atteinte d'un bassin d'attraction plus favorable, évitant au modèle de rester prisonnier d'un mauvais minimum local, comme c'est le cas avec la simple initialisation aléatoire. Ainsi, en plus de l'utilisation d'un critère supervisé pertinent à la tâche, la clef d'une bonne généralisation dépend de l'utilisation d'un critère non-supervisé guidant l'apprentissage à chacune des couches. En un sens, cette technique se rapproche de l'apprentissage semi-supervisé, à la différence qu'au lieu de se servir d'exemples supplémentaires non étiquetés, les exemples sont d'abord utilisés sans

leur cible durant une première phase de pré-entraînement.

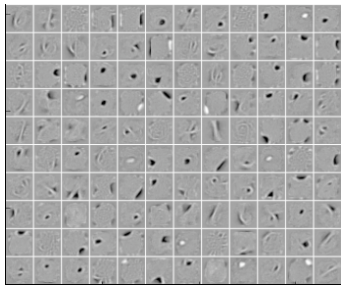
Ensuite, [45] ont révélé que l'usage du module DAE pour l'initialisation du réseau donnait, pour la majorité des ensembles de données testés, des performances supérieures à son prédécesseur AE, et s'avérant meilleures ou similaires à celles du SVM et du DBN. De plus, c'est en observant les filtres appris du premier DAE d'une architecture profonde que [45] ont pu montrer visuellement et qualitativement l'effet bénéfique du débruitage (voir l'image 2.4). On observe que, sans bruit, les filtres observés sont locaux et peu intéressants. Tandis que l'ajout de bruit de type (MN) permet l'apparition de filtres moins locaux, notamment des détecteurs de petites *gouttes*, des détecteurs de traits, et pour un niveau de bruit élevé, certains détecteurs de caractères complets. Ce dernier résultat n'est pas surprenant considérant que l'augmentation du niveau de bruit augmente conséquemment la corruption locale et ne laisse donc aucun autre choix au DAE que de se tourner vers l'information davantage globale, notamment en apprenant des filtres s'activant pour davantage de dimensions.



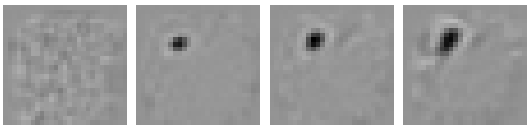
(a) No corruption



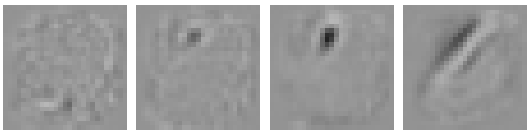
(b) 25% corruption



(c) 50% corruption



(d) Neuron A (0%, 10%, 20%, 50% corruptions)



(e) Neuron B (0%, 10%, 20%, 50% corruptions)

Figure 2.4 – **Filtres obtenus après l'entraînement d'un DAE sur** *basic***.** (a-c) montrent des exemples de filtres obtenus selon une fraction ν croissante de bruit MN appliqué à l'entrée. (d) et (e) montrent les filtres de deux neurones cachés, selon une proportion de bruit croissante. On remarque qu'avec l'absence de bruit, les filtres sont locaux et ne présentent pas de structures particulièrement intéressantes, tandis que l'augmentation de la proportion de bruit fait apparaître au sein des filtres des différences intéressantes et des structures nécessitant un plus grand nombre de dimensions.

CHAPITRE 3

NOUVELLES STRATÉGIES DE PRÉ-ENTRAÎNEMENT

Ayant constaté le succès du pré-entraînement par débruitage qu'implique l'architecture SDAE [45], nous avons voulu d'une part, vérifier l'effet de l'inclusion d'un bruitage à l'intérieur du pré-entraînement avec empilement de RBM, et d'autre part, vérifier la performance de nouvelles stratégies de pré-entraînement s'inspirant du SDAE. Dans ce chapitre, nous détaillons ces nouvelles variantes de pré-entraînement dont nous évaluerons, au chapitre suivant, la qualité des représentations sur-complètes apprises au niveau de la première couche cachée de même que l'efficacité en terme d'initialisation des paramètres du réseau pour la tâche supervisée de classification.

3.1 Variantes de pré-entraînement associées à la machine de Boltzmann restreinte (RBM)

Comme l'inclusion de bruit au niveau du pré-entraînement d'une architecture composée d'auto-encodeurs empilés a permis une nette amélioration des performances sur la tâche supervisée [45], nous avons voulu vérifier s'il en était de même pour le cas du pré-entraînement couche par couche d'une architecture composée de machines de Boltzmann restreintes. Deux variantes sont étudiées. Chacune d'elles consiste en l'utilisation d'une version partiellement corrompue du vecteur d'entrée x^0, mais à deux moments différents lors de l'entraînement du RBM. Dans un instant, nous verrons à quels moments précisément la version corrompue, $\tilde{x}^0$, est employée pour chacun des deux cas. La corruption est la même que celle utilisée par [45], c'est-à-dire qu'un processus de corruption q_ν met à zéro une fraction ν choisie au hasard des composantes d'entrées.

3.1.1 Détails du module RBM$_{fprop}$

Pour la première variante expérimentée, que l'on nomme RBM$_{fprop}$, une version partiellement corrompue du vecteur d'entrée, $\tilde{x}^0$, est utilisée pour les phases positive et

négative de propagation de l'information, alors que l'entrée originale, x, est employée lors de la mise à jour des paramètres. Ainsi, les équations liées à la phase positive deviennent :

$$x^0 \leftarrow x \tag{3.1}$$
$$\tilde{x}^0 \sim q_v(x^0) \tag{3.2}$$
$$\hat{h}^0 \leftarrow s(b + W\tilde{x}^0) \tag{3.3}$$

tandis que la phase négative est maintenant définie comme suit :

$$h^0 \sim p(h|\tilde{x}^0) \tag{3.4}$$
$$x^1 \sim p(x|h^0) \tag{3.5}$$
$$\hat{h}^1 \leftarrow \text{sigmoide}(b + Wx^1), \tag{3.6}$$

La mise à jour, elle, reste la même que celle d'un RBM traditionnel, c'est-à-dire définie par les équations retrouvées en 2.11, que nous reproduisons ici.

$$\begin{aligned} W^i &\leftarrow W^i + \eta * (\hat{h}^0(x^0)^T - \hat{h}^1(x^1)^T) \\ b^i &\leftarrow b^i + \eta * (\hat{h}^0 - \hat{h}^1) \\ c^i &\leftarrow c^i + \eta * (x^0 - x^1) \end{aligned}$$

3.1.2 Détails du module RBM$_{update}$

La seconde variante étudiée, RBM$_{update}$, est caractérisée par l'utilisation de $\tilde{x}^0$ seulement lors de la mise à jour des paramètres. Ainsi, les phases de propagation positive et négative n'en sont pas affectées et conservent les mêmes équations que celles vues en 2.8,2.6. La mise à jour est maintenant définie comme suit :

$$\tilde{x}^0 \sim q_v(x^0) \tag{3.7}$$

$$W^i \leftarrow W^i + \eta * (\hat{h}^0(\tilde{x}^0)^T - \hat{h}^1(x^1)^T) \tag{3.8}$$

$$b^i \leftarrow b^i + \eta * (\hat{h}^0 - \hat{h}^1) \tag{3.9}$$

$$c^i \leftarrow c^i + \eta * (\tilde{x}^0 - x^1) \tag{3.10}$$

3.1.3 Nouvelles architectures résultantes

Ces deux variantes du module RBM, soit RBM$_{fprop}$ ou RBM$_{update}$, lorsqu'empilées et entraînées l'une à la suite de l'autre, résultent en l'initialisation des architectures que nous nommerons DBN$_{fprop}$ et DBN$_{update}$ respectivement. Au chapitre suivant, nous verrons la qualité des filtres appris et la performance de généralisation obtenue via l'emploi de ces variantes bruitées de RBM pour le pré-entraînement d'un réseau profond, en comparaison du DBN pré-entraîné avec des RBM traditionnelles.

3.2 Variantes de pré-entraînement inspirées de l'auto-encodeur débruiteur (DAE)

Pour les variantes d'auto-encodeur débruiteur que nous allons maintenant proposer, les paramètres sont mis à jour de la même manière que dans le cas de l'AE (section 2.3.2) et du DAE (section 2.3.3), soit à l'aide d'une descente de gradient stochastique sur la perte (équations 2.16,2.17) qui est fonction de x et de sa "reconstruction" z. La différence d'un module à l'autre, est la nature même de x et de z.

3.2.1 Auto-encodeur comblant les données manquantes (MAE)

Bien que les DAE avec la corruption de type MN proposé dans [45] aient été motivés par l'idée d'apprendre à compléter des données "manquantes" introduites artificiellement, l'opération de débruitage ne correspond pas exactement à la complétion de données manquantes. En effet, forcer la valeur d'une composante à une valeur prédéfinie n'est pas tout à fait la même chose que d'indiquer que cette composante est manquante.

En particulier un DAE ne fait pas la différence entre une composante d'entrée non corrompue, mais de valeur égale au masque de corruption appliqué, et une dimension d'entrée qui a bel et bien été masquée. En effet, si l'on considère un vecteur d'entrée de composantes $\in [0,1]$, l'influence d'une composante de valeur égale au masque utilisé, sur un neurone i de la couche de représentation (couche cachée) est la même que celle d'une composante qui a été masquée. En effet, sachant que l'influence d'une composante d'entrée x_k sur une unité h_i de la couche supérieure correspond à $W_{i,k}x_k$, on obtient les influences suivantes selon la nature de x_k :

$$Effet = \begin{cases} W_{i,k} & \text{si } x_k = 1 \text{ ou si } x_k \text{ est masqué à 1} \\ 0 & \text{si } x_k = 0 \text{ ou si } x_k \text{ est masqué à 0} \\ W_{i,k} * x_k & \text{autrement} \end{cases} \tag{3.11}$$

L'objectif du module MAE (*deMissing Auto-Encoder*) est donc d'intégrer un état supplémentaire, soit celui des *données ou caractéristiques manquantes* (état MD), indiquant que la valeur originale a été perdue lors d'un processus de corruption. L'espoir est que l'ajout de l'état MD comme information supplémentaire puisse permettre à l'auto-encodeur d'apprendre une meilleure représentation intermédiaire, menant en fin de compte à une meilleure performance sur la tâche supervisée.

Il existe différentes manières d'inclure l'état MD. Deux d'entre elles seront vues et testées. Toutes deux impliquent les étapes d'apprentissage qui suivent : une première transformation $r(x)$ est appliquée au vecteur d'entrée x, de dimension d, qui résulte en un vecteur u de dimension deux fois celle de x. La moitié des composantes de u correspond à x, tandis que l'autre moitié contient de l'information qui permettra l'intégration de l'état MD. Une fraction ν des caractéristiques de x, choisie au hasard, sont étiquetées comme manquantes. Le vecteur u est dès lors modifié en conséquence, advenant le vecteur $\widetilde{u}$. Nous reviendrons sous peu sur la nature exacte des vecteurs u et $\widetilde{u}$. Ensuite, le vecteur partiellement corrompu $\widetilde{u}$ est encodé en une représentation, de la même façon qu'avec les AE et DAE, puis décodé de manière à obtenir le vecteur z', de dimension deux fois celle de l'entrée que le MAE doit être en mesure de reconstruire. Finalement,

le vecteur sortant z, de dimension d résulte d'une transformation qui est propre à la nature de u. La figure 3.1(a) montre ces étapes d'apprentissage. Tout comme les modules vus précédemment, une optimisation des paramètres est effectuée à l'aide d'une descente de gradient stochastique sur l'entropie croisée entre x et z (qui est l'erreur de reconstruction utilisée pour des entrées binaires).

C'est la nature de u et celle de $\widetilde{u}$ qui viendront différencier la méthode utilisée pour la considération de l'état MD. Nous nommerons la première méthode *binomial missing* (MAE$_{BM}$) et la seconde ; *one if missing* (MAE$_{OM}$).

3.2.1.1 Détails du module MAE$_{BM}$

Une condition à l'utilisation de MAE$_{BM}$ est la nature binaire ou continue dans l'intervalle $[0,1]$ des composantes d'entrée. Le vecteur u est dès lors composé d'une part, des composantes de x_k, d'autres part, de leur valeurs *"opposées"* respectives, soit $1-x_k$. Lors de la phase de corruption, on étiquette la donnée x_k comme manquante en mettant à zéro le couple d'unité associé dans le vecteur u. Ainsi, on remarque qu'une caractéristique x_k de l'entrée possèdera nécessairement une influence différente sur un neurone i de la représentation, selon qu'elle est ou non manquante, à moins que les poids $W_{i,2k}$ et/ou $W_{i,2k+1}$ ne soient nuls :

$$Effet = \begin{cases} 0 & \text{si } x_k \text{ est manquant} \\ W_{i,2k} & \text{si } x_k = 1 \\ W_{i,2k+1} & \text{si } x_k = 0 \\ W_{i,2k} * x_k + W_{i,2k+1} * (1 - x_k) & \text{autrement} \end{cases} \tag{3.12}$$

Pour le passage du vecteur z' au vecteur z de dimension d, nous considérons la contrainte connue suivante : deux à deux, les neurones du vecteur u somment à 1 et ce sont les neurones d'indice paire qui forment le vecteur d'entrée x (considérant des indices débutant à zéro). Nous imposons donc cette contrainte au vecteur z' pour ensuite considérer ses neurones d'indice pair pour l'obtention du vecteur de reconstruction z.

Ainsi, la valeur d'une composante k du vecteur z est le résultat du softmax suivant : $z_k = \frac{\mathbf{e}^{z'_{2k}}}{\mathbf{e}^{z'_{2k}}+\mathbf{e}^{z'_{2k+1}}}$. Or, par souci de simplicité, nous allons considérer l'identité qui suit :

$$z_k = \frac{\mathbf{e}^{z'_{2k}}}{\mathbf{e}^{z'_{2k}} + \mathbf{e}^{z'_{2k+1}}} = \frac{\mathbf{e}^{z'_{2k}}}{\mathbf{e}^{z'_{2k}} + \mathbf{e}^{z'_{2k+1}}} * \frac{\mathbf{e}^{z'_{2k}}}{\mathbf{e}^{z'_{2k}}} \tag{3.13}$$

$$= \frac{1}{1 + \mathbf{e}^{z'_{2k+1} - z'_{2k}}} \tag{3.14}$$

$$= \text{sigmoide}(z'_{2k} - z'_{2k+1}) \tag{3.15}$$

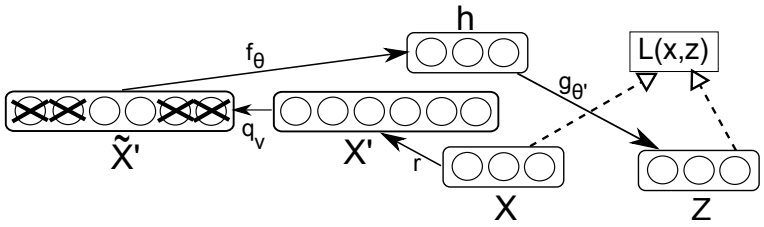
Ainsi, pour la méthode MAE_{BM}, on obtient la reconstruction z en soustrayant chaque composante de z' d'indice impair à son voisin d'indice inférieur, puis en opérant une sigmoïde sur chacun des résultats.

3.2.1.2 Détails du module MAE_{OM}

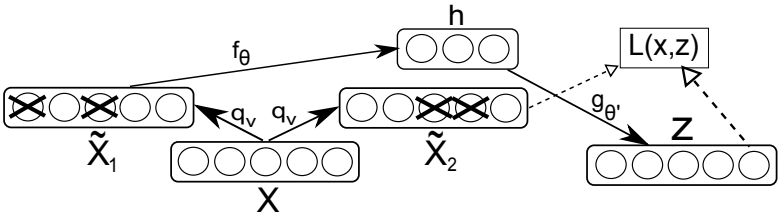
En ce qui a trait à la méthode MAE_{OM}, le vecteur u est formé avec les composantes de x accompagnées d'unités de valeurs nulles. Puis, tel que son nom l'indique (*one if missing*), le processus de corruption mettra à 1 une fraction ν de ces unités originairement nulles et à zéro les composantes précédentes de manière à indiquer que ces dernières ont été étiquetées comme manquantes. En d'autres mots, l'unité d'indice impair sert à indiquer si oui (1) ou non (0) la composante d'indice inférieur possède une valeur nulle parce qu'elle est manquante. Le vecteur de reconstruction z est dès lors obtenu en ne conservant que les unités d'indice pair de z'.

Ainsi, tout comme la méthode précédente, on remarque que l'influence d'une caractéristique d'entrée, x_k, sur un neurone i de la représentation sera différentes selon que x_k est ou non manquante. Par ailleurs, on observe que du moment qu'il n'y a aucune donnée manquante, cette méthode revient équivalente à l'utilisation d'un AE :

$$Effet = \begin{cases} W_{i,2k+1} & \text{si } x_k \text{ est manquant} \\ W_{i,2k} & \text{si } x_k = 1 \\ 0 & \text{si } x_k = 0 \\ W_{i,2k} * x_k & \text{autrement} \end{cases} \tag{3.16}$$



(a) Auto-encodeur comblant les données manquantes (MAE)



(b) Auto-encodeur rebruiteur (RAE)

Figure 3.1 – **Les étapes d'entraînement d'un auto-encodeur comblant les données manquantes (MAE) et d'un auto-encodeur rebruiteur (RAE).** Dans les deux cas, l'entraînement consiste à optimiser, jusqu'à l'atteinte d'un critère d'arrêt, les paramètres θ et θ' de manière à minimiser la moyenne de la perte choisie entre x, pour MAE, ou $\widetilde{x}_2$, pour RAE, et la sortie du module, z, sur l'ensemble d'apprentissage. Dans le cas du MAE, un vecteur de dimension deux fois celle de l'entrée est obtenu, $x' = r(x)$, via une transformation qui ajoute de l'information pertinente à l'étiquetage des données manquantes. Le processus de corruption q_ν prend en entrée x', sélectionne au hasard une fraction ν des composantes de ce dernier, les étiquette comme manquantes puis obtient en sortie le vecteur $\widetilde{x}'$. Ensuite, la fonction $f_\theta(\widetilde{x}')$ applique une transformation non-linéaire de $\widetilde{x}'$ résultant en une représentation h, et finalement, $z = g_{\theta'}(h)$ est le vecteur que l'on voudra le plus semblable possible à l'entrée non corrompue x. En ce qui concerne le module RAE, deux versions corrompues, $\widetilde{x}_1$ et $\widetilde{x}_2$, sont obtenues à partir de l'entrée x. La première version servira à l'obtention de la représentation h, tandis que la seconde sera le vecteur dont on voudra que z se rapproche du mieux possible.

3.2.2 Auto-encodeur rebruiteur (RAE)

Finalement, un dernier module a fait l'objet d'une étude, il s'agit de l'auto-encodeur rebruiteur (RAE). Il s'avère très semblable au DAE qui apprend à reconstruire du mieux possible un exemple x à partir d'une version bruitée de ce dernier, soit $\widetilde{x}$. La variante ci-proposée, consiste toujours à alimenter le module d'une version bruitée de l'entrée, que l'on nommera $\widetilde{x}_1$, mais cette fois-ci, on désire forcer la sortie du module à se rapprocher non pas de l'exemple original, mais d'une seconde version bruitée de ce dernier : $\widetilde{x}_2$.

La motivation pour cette variante est principalement liée à une hypothèse biologique quant à la corruption des données. En effet, il semblerait davantage naturel et biologiquement plausible, si un bruit physique vient corrompre la donnée d'entrée, que la cible de reconstruction soit similairement corrompue.[1] La figure 3.1(b) montre les étapes propres à ce module.

3.2.3 Une extension à DAE et MAE : appliquer une emphase sur les composantes corrompues

Par *composantes corrompues*, on entend les données bruitées (dans un DAE avec un bruit MN par exemple) de même que les données étiquetées comme manquantes (explicitement dans un MAE). La présente extension consiste à imposer une emphase sur la reconstruction des données corrompues lors du calcul de "l'erreur de reconstruction" entre x et z. Par exemple, au lieu d'utiliser l'entropie croisée définie en 2.16, on appliquera la perte suivante :

$$L(x,z') = -\sum_{k=1}^{d} [(m_k\alpha + (1-m_k)\gamma)(x_k \log(z_k) + (1-x_k)\log(1-z_k))] \tag{3.17}$$

où $m_k \in \{0,1\}$ indique si la composante x_k a été corrompue (1) ou non (0) lors du processus de corruption, tandis que $\alpha \in [0,1]$ et $\gamma \in [0,1]$ sont les poids attribués à la re-

[1] Remarque : pour un bruit additif gaussien et une erreur de reconstruction quadratique, la meilleure reconstruction possible sera l'espérance des exemples corrompus c'est-à-dire l'exemple d'origine non corrompu. Mais les choses ne sont pas aussi simples avec d'autres types de corruption ou avec une erreur de reconstruction différente telle que l'entropie croisée.

construction d'une composante corrompue et non corrompue, respectivement. Ainsi, la combinaison $\alpha = 1; \gamma = 0$ implique une emphase entière sur la reconstruction des données corrompues. En employant une telle emphase, on force le module à concentrer son apprentissage principalement, ou en totalité, sur la reconstruction des données corrompues. En fonctionnant de la sorte, on espère obtenir un module davantage robuste aux entrées corrompues et qui sera en mesure d'extraire de l'entrée des régularités facilitant la tâche supervisée.

3.2.4 Algorithmes d'entraînement de réseaux profonds correspondant à ces nouvelles variantes

Dans cette section, nous avons détaillé les nouveaux modules de pré-entraînement MAE_{OM}, MAE_{BM} et RAE, en plus d'ajouter une extension à DAE et MAE, soit l'application d'une emphase au niveau de la reconstruction des composantes corrompues, ou manquantes. L'empilement (*Stacking*) et l'entraînement successif de ces modules, pour l'initialisation des paramètres d'un réseau de neurones profond, suivi d'un entraînement supervisé global, donnent les algorithmes que nous nommerons respectivement SMAE_{OM}, SMAE_{BM} et SRAE. Nous évaluerons au chapitre 4 leurs performances de classification et procéderons à une évaluation visuelle qualitative des paramètres appris après l'entraînement de la première couche.

CHAPITRE 4

EXPÉRIMENTATION ET RÉSULTATS

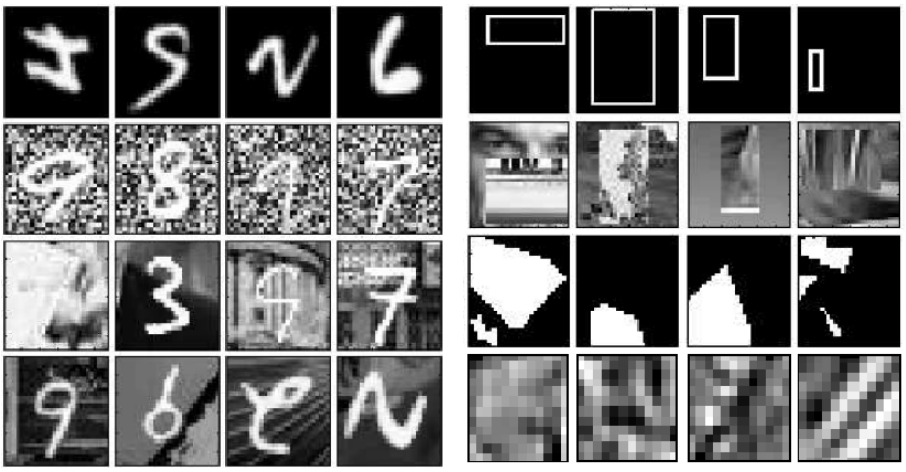
Dans ce chapitre, nous allons explorer différentes facettes de l'architecture SDAE qui permettront de mieux comprendre le processus derrière ce type de pré-entraînement. Nous expérimenterons ensuite les nouvelles stratégies de pré-entraînement décrites au chapitre précédent.

4.1 Ensembles de données

Les algorithmes de classification ont été testés dans un premier temps sur *MNIST*, un jeu de données naturelles bien connu composé des chiffres blancs de 0 à 9 écrits à la main sur fond noir, *basic*, un sous-ensemble du même jeu de donnée, puis quatre variantes de ce dernier tirés de [28]. Les variantes incluent une rotation des chiffres d'un angle choisi uniformément $\in [0, 2\pi]$ (*rot*), un arrière-fond composé de pixels aux valeurs générées aléatoirement $\in [0, 255]$ (*bg-rand*), une image naturelle en noire et blanc en guise d'arrière-fond (*bg-img*) et finalement, la combinaison d'une rotation des chiffres et d'une image naturelle comme arrière-fond (*rot-bg-img*). Trois autres jeux tirés de [28] ont été employés. Il s'agit de jeux artificiels consistant d'une part en des images de rectangles blancs sur fond noir (*rect*) ou avec une image naturelle en arrière-fond (*rect-img*) que l'on peut classifier selon que la hauteur ou la largeur est prédominante, puis d'autre part, en des images contenant une forme convexe ou concave (*convex*). Tout ces problèmes constituent un défi, considérant la haute dimension de leurs entrées, soit 784 (images de $28 \times 28 = 784$ pixels), et sachant que les humains excellent à ces tâches alors que la performance des algorithmes d'apprentissage se retrouve loin derrière. Deux jeux de données supplémentaires ont été utilisés lors de la présente étude. Le premier consiste en une variante du jeu de donnée *tzan-MPC* [9], contenant 10000 clips auditifs de trois secondes, également séparés en 10 genres musicaux : blues, classique, country, disco, hiphop, pop, jazz, métal, reggae et rock. Un exemple de l'ensemble est formé de 592 ca-

ractéristiques *Mel Phon Coefficient* (MPC), une formulation simplifiée de *Mel-frequency cepstral coefficients* (MFCCs), donnant lieu à une meilleure performance de classification [9]. Le dernier jeu de données auquel nous ferons référence est *olsh-12x12* composé de parties d'images naturelles utilisé par Olshausen [36], et dont la dimension a été réduite à 12 par 12.

À l'exception d'*olsh-12x12*, les jeux de données ont servi à un problème de classification et ont été divisés en trois ensembles : un ensemble d'entraînement, de validation, puis de test. À noter que dans le cas de *tzan-MPC*, étant donné son faible nombre d'exemples, une validation croisée a été utilisée et sera détaillée dans la prochaine section. Les tailles des différents ensembles ainsi que toute l'information résumée sur les jeux de données se trouvent à la table 4.1. Finalement, la figure 4.1 montre un échantillon de chacun des jeux de données à l'exception de *tzan-MPC*.



(a) Variantes de MNIST (b) Rectangle, Convexe et Olshausen

Figure 4.1 – **Échantillons des jeux de données.** Dans la moitié gauche (a) : des échantillons des variantes de *MNIST*, soit de haut en bas : *rot*, *bg-rand*, *bg-img* et *rot-bg-img*. Dans la moitié droite (b) : trois jeux de données artificielles : *rect*, *rect-img* et *convex* et finalement, les parties d'images naturelles de *olsh-12x12*.

4.2 Types de bruit

Le type de bruit présenté par [45] qui a été utilisé pour l'entraînement du DAE, est un bruit caractérisé par l'application d'un masque à zéro (MN : *Masking Noise avec masque à zéro*), c'est-à-dire qu'une certaine fraction ν des composantes de x (choisies aléatoirement pour chaque exemple), est mise à zéro. On peut interpréter ce type de bruit comme l'élimination de l'information associée à une composante considérée comme manquante ou encore l'attribution d'une valeur par défaut. Dans ce contexte, on peut imaginer le

Tableau 4.1 – **Information sur les jeux de données.** Tableau comprenant la description des différents problèmes sur lesquels nous avons testé nos algorithmes d'apprentissage, le pré-traitement appliqué, leur dimension, le nombre de classes (*Nb. Cl.*) et le nombre d'exemples utilisés pour l'ensemble d'entraînement (*train*), de validation (*valid*) et de test respectivement. À noter que dans le cas de *tzan-MPC*, une validation croisée a été effectuée sur les 10000 exemples disponibles. Pour *tzan-MPC* et *olsh-12x12*, un pré-traitement à été appliqué de manière à obtenir pour chacune des dimensions, une moyenne μ et un écart-type σ particuliers. Pour *MNIST* et ses variantes, les composantes ont été ramenées à des valeurs entre 0 et 1. Puis concernant les trois jeux artificiels, *rect* et *convex* contiennent des images avec pixels de valeurs binaires, tandis que les images de *rect-img* ont des composantes de valeur située entre 0 et 1.

Données	**Description**	**Pré traitement**	**Dim**	**Nb. Cl.**	**Train-Valid-Test**
MNIST	MNIST original	valeurs $\in[0,1]$	784	10	50000-10000-10000
basic	sous ensemble de *MNIST*	valeurs $\in[0,1]$	784	10	10000-2000-50000
rot	*basic* + rotation	valeurs $\in[0,1]$	784	10	10000-2000-50000
bg-rand	*basic* + arrière-fond bruité	valeurs $\in[0,1]$	784	10	10000-2000-50000
bg-img	*basic* + img en arrière-fond	valeurs $\in[0,1]$	784	10	10000-2000-50000
rot-bg-img	*bg-img* + rotation	valeurs $\in[0,1]$	784	10	10000-2000-50000
rect	rectangle large / long	valeurs $\in\{0,1\}$	784	2	1000-200-50000
rect-img	*rect* + img en arrière-fond	valeurs $\in[0,1]$	784	2	10000-2000-50000
convex	forme convexe / concave	valeurs $\in\{0,1\}$	784	2	6000-2000-50000
tzan-MPC	séquence musical	μ=0,σ=1	592	10	10000
olsh-12X12	parties d'images naturelles	*whitened* 20x20→12x12 μ=0,σ=0.27	144	N-A	100000

rôle du DAE comme étant celui de retrouver l'information perdue. Par exemple, pour l'image représentant un chiffre blanc sur fond noir, le bruitage forme des trous dans le chiffre et l'apprentissage non-supervisé a alors comme objectif de les reboucher. Ce type de bruit est aussi motivé par le fait qu'une valeur d'entrée de 0 n'entre pas en compte dans le calcul des activations des neurones. Mais ce type de corruption, avec mise à 0, paraît arbitraire notamment en ce que l'effet de corruption perçu va être très différent si l'on a préalablement inversé les valeurs des pixels (le blanc et le noir). Aussi, nous allons explorer l'utilisation de deux autres types de bruits mieux justifiés. Le premier consiste en un bruit s'inspirant du masque à zéro, soit le bruit *poivre et sel* (PS), c'est-à-dire qu'une fraction ν des composantes de x (choisies aléatoirement pour chaque exemple) se verra attribuer leur valeur minimale (*poivre*) ou maximale (*sel*), selon une probabilité égale d'obtenir l'une ou l'autre des valeurs. Ce type de bruit se prête bien aux données de domaine que l'on peut interpréter comme binaire, telles que les images en noir et blanc, ou encore aux représentations issues d'une non-linéarité de type sigmoïde. On utilisera dès lors une valeur *poivre* égale à 0 et une valeur *sel* égale à 1. Le second type de bruit dont nous examinerons l'influence est le bruit additif gaussien (GS) : $\widetilde{x}|x \sim \mathcal{N}(x, \sigma^2 I)$. Il s'agit tout simplement d'ajouter à toutes les composantes d'entrée un bruit gaussien d'écart type σ. Ce type de bruit est plus approprié pour les entrées de domaine réel, telles que les images naturelles.

Il faut souligner que pour le cas d'images en noir et blanc, une même fraction ν de composantes bruitées résultera en une corruption plus accrue si l'on utilise le bruit PS plutôt que le bruit MN, et ce étant donné que le bruit *masque à zéro* ne vient affecter que les pixels blancs tandis que le bruit *poivre et sel* viendra corrompre autant les pixels noirs que les pixels blancs. Par ailleurs, alors que le bruit GS corrompt toutes les composantes, les bruits MN et PS ne corrompent qu'une fraction de ces derniers, laissant intactes les autres. Dans le contexte d'utilisation des bruits MN ou PS, le débruitage n'est possible que s'il existe une dépendance entre les différentes dimensions. On s'attend donc à ce que le pré-entraînement soit en mesure de déceler ces dépendances de manière à pouvoir remplacer l'information absente. Ces types de bruits perdraient donc probablement leur sens en présence de données de faibles dimensions exemptes de dépendances.

4.3 Protocole général

Les différentes architectures considérées ici consistent en une couche visible (d'entrée), une à trois couches cachées puis une couche de sortie. Les couches cachées possèdent toutes un nombre égal d'unités. Chaque couche cachée effectue une transformation affine de l'entrée qu'elle reçoit suivie d'une non-linéarité de type sigmoïde. Lors du pré-entraînement par les diverses variantes d'auto-encodeurs, la reconstruction des couches cachées est de type *Bernoulli*, c'est-à-dire qu'on utilise là aussi une non-linéarité de type sigmoïde avec un coût de reconstruction d'entropie croisée. Ce même type de reconstruction est utilisé au niveau de la première couche visible pour les jeux de données dont les entrées sont binaires ou quasi binaires (soit *MNIST* et ses variantes, ainsi que les trois jeux artificiels *rect*, *rect-img* et *convex*). Pour les jeux de données *tzan-MPC* et *olsh-12x12*, les entrées ont davantage une nature réelle que binaire, et nous utilisons donc une couche de reconstruction de type *gaussienne*, sans non-linéarité et optimisant un coût de reconstruction quadratique.

Un dernier type de couche a été utilisé, mais seulement pour un cas particulier, en tant que couche visible à l'architecture DBN sur les ensembles de données *MNIST* et ses variantes. Il s'agit de la couche *exponentielle tronquée*. Nous suivons en ceci les travaux de [29] qui ont dévoilé une différence significative au niveau des filtres appris et une amélioration de la performance de classification sur l'ensemble *bg-img*, lorsque comparé à l'utilisation d'une couche d'entrée *binaire (Bernoulli)*. Des résultats supplémentaires obtenus par l'utilisation d'une couche d'entrée de type *gaussienne* ou *exponentielle tronquée* se retrouvent dans [7].

Finalement, la non-linéarité employée à la couche de sortie, est la *softmax*, permettant l'obtention d'une probabilité pour chacune des classes que l'exemple en cours fasse bel et bien partie de cette classe.

Aussi, à moins d'être précisé, les matrices d'encodage et de décodage utilisées pour chacun des auto-encodeurs sont liées l'une à l'autre, c'est-à-dire qu'une seule matrice est utilisée, soit la matrice originale lors de l'encodage, et la matrice transposée lors du décodage. Il s'en suit qu'une mise à jour de la matrice lors du décodage entraîne né-

cessairement la modification de la matrice utilisée lors de l'encodage et vice-versa. La recherche des hyper-paramètres optimaux a été conduite de manière similaire à [28], c'est-à-dire que pour une architecture fixée, nous avons procédé au choix des meilleures valeurs des différents hyper-paramètres (parmi une large grille de valeurs) selon la performance sur l'ensemble de validation. Les hyper-paramètres concernés dans la majorité des expériences sont les suivants : nombre de neurones par couches cachées, le taux d'apprentissage pour la phase non-supervisée et pour la phase supervisée, le nombre de mises à jour des paramètres lors de la phase de pré-entraînement couche par couche, la fraction de composantes d'entrées qui seront bruitées ou étiquetées comme étant manquantes de même que, si utilisé, l'écart-type du bruit gaussien appliqué sur toutes les composantes. Dans tous les cas, les taux d'apprentissage pour la phase supervisée et non-supervisée sont constants tout au long de la phase. Aussi, à moins d'être précisé, aucune régularisation de type *weight-decay* n'a été appliquée. Dans le cas de l'usage du bruit PS, nous avons utilisé, pour toutes les expériences, une probabilité à 50% qu'une composante soit masquée par la valeur du *sel* plutôt que par celle du *poivre*. Finalement, l'usage d'une emphase au niveau de la reconstruction des données corrompues, nécessite l'emploi de deux hyper-paramètres supplémentaires, $\in [0,1]$, soit le poids accordé, lors de la reconstruction, aux composantes corrompues et aux composantes non corrompues respectivement. Dans le cadre de l'étude actuelle, nous avons utilisé un poids de 1 pour les composantes corrompues et avons testé trois valeurs de poids sur les composantes non corrompues, soit $\in \{0, 0.5, 1\}$ de sorte à obtenir une *grosse*, une *moyenne* emphase ou *aucune* emphase sur la reconstruction des composantes corrompues. Chaque modèle inclut l'arrêt prématuré lors de l'entraînement supervisé, plus précisément, l'entraînement est arrêté si l'erreur sur l'ensemble de validation ne s'est pas améliorée à l'intérieur de cinq époques consécutives. Aucun critère d'arrêt de ce genre n'est utilisé dans la phase de pré-entraînement, on utilise plutôt l'hyper-paramètre *nEpoq* afin de définir le nombre de passages au travers de l'ensemble d'entraînement.

Les intervalles de valeurs testées pour chacun des hyper-paramètres se trouvent dans la table 4.2.

Pour chaque jeu de données, le modèle avec la meilleure performance obtenue sur

Hyper-paramètre	Valeur
nHLay	{1,2,3}
nHUnit	{1000,2000,3000}
lRate	{5x10^{-6},5x10^{-5},5x10^{-4},5x10^{-3},5x10^{-2},10^{-1}}
lRateSup	{0.0005,0.005,0.05,0.1,0.15,0.2}
nEpoq	{5,10,50,100,125,150,200,300}
fmi	{0.1,0.25,0.4}%
gsStdDev	{0.05,0.1,0.15,0.3,0.5}
wCor / wUncor	1 / {0,0.5,1}

Tableau 4.2 – **Valeurs testées pour différents hyper-paramètres**. Dans l'ordre, en commençant par le haut : *(nHLay)* nombre de couches cachées, *(nHUnit)* nombre d'unités par couche cachée (même nombre pour chacune des couches), *(lRate)* pas d'apprentissage pour la phase non-supervisée, *(lRateSup)* pas d'apprentissage pour la phase supervisée, *(nEpoq)* nombre de passages sur l'ensemble d'entraînement pour la phase non-supervisée, *(fmi)* fraction d'entrée bruitée ou étiquetée manquante, *(gsStdDev)* écart-type utilisé pour le bruit gaussien, *(wCor)* poids attribué à la reconstruction d'une composante d'entrée corrompue, *(wUncor)* poids attribué à la reconstruction d'une composante d'entrée non corrompue.

l'ensemble de validation a été retenu. Les intervalles de confiance à 95% sur la moyenne des erreurs de test ont été calculés selon l'équation suivante : $\hat{\mu} \pm z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}} = \hat{\mu} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{\mu}(1-\hat{\mu})}{n}}$, où $\hat{\mu}$ est la moyenne des erreurs de test, $\alpha = 0.05$, n égale le nombre d'exemples de test tandis que $z_{1-\alpha/2}$ est l'inverse de la distribution cumulative d'une gaussienne N(0,1) à $1-\alpha/2$. Sachant les erreurs binomiales (1 si erreur de classification, 0 sinon), voici la preuve que l'écart-type des erreurs de test (σ) est effectivement

égale à $\sqrt{\hat{\mu}(1-\hat{\mu})}$:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{N}(\mu-\hat{\mu})^2} \tag{4.1}$$

$$= \sqrt{\frac{1}{n}[\sum_{\mu=0}(\mu-\hat{\mu})^2 + \sum_{\mu=1}(\mu-\hat{\mu})^2]} \tag{4.2}$$

$$= \sqrt{\frac{1}{n}[n\hat{\mu}^2(1-\hat{\mu}) + n\hat{\mu}(1-\hat{\mu})^2]} \tag{4.3}$$

$$= \sqrt{\hat{\mu}(1-\hat{\mu})[\hat{\mu}+1-\hat{\mu}]} \tag{4.4}$$

$$= \sqrt{\hat{\mu}(1-\hat{\mu})} \tag{4.5}$$

Dans le cas de jeu de données *tzan-MPC*, étant donné que celui-ci ne comporte qu'un total de 10000 exemples, une validation croisée a été effectuée telle que les 8000 premiers exemples ont d'abord servi à l'entraînement, les 1000 suivants à la validation et les 1000 derniers au test. Cette procédure a été répétée à dix reprises, toujours en décalant de 1000 le début des ensembles d'entraînement, de validation et de test. Pour cet ensemble de données particulièrement, au lieu d'utiliser l'intervalle de confiance mentionné ci-haut, nous avons calculé l'écart-type obtenu sur les dix moyennes d'erreur de test obtenues.

4.4 Validation de la technique de pré-entraînement

Les travaux de [45] comparaient la performance d'un réseau profond à trois couches cachées initialisé par un empilement d'auto-encodeurs débruiteurs (SDAE-3) aux algorithmes suivants :

- SVM_{rbf}, un SVM à noyau gaussien
- SVM_{poly}, un SVM à noyau polynomial
- DBN-3, un réseau à trois couches cachées initialisé avec un empilement de RBM
- SAE-3, un réseau à trois couches cachées initialisé avec un empilement d'auto-encodeurs ordinaires

Dans un premier temps, nous avons reproduit une partie de ces résultats et les avons complétés en obtenant les performances sur les ensembles de données *MNIST* et *tzan-MPC* (voir les résultats situés dans la table 4.3). À noter que le modèle SDAE-3 a été entraîné avec le bruit masque à zéro pour tous les ensembles de données, sauf pour *tzan-MPC* où la notion de *trous dans l'image* n'est plus adéquate. Ainsi pour ce dernier jeu de données, nous avons employé le bruit gaussien, approprié pour les entrées de domaine réel. Il ressort notamment du tableau de résultats que le SDAE obtient sur tous les ensembles de données, sauf *convex*, une meilleure performance de généralisation que son prédécesseur, le SAE. De plus, pour toutes les tâches, sa performance est équivalente ou supérieure aux autres modèles, sauf en ce qui concerne *bg-rand*, où DBN-3 l'emporte avec une erreur de test de 6.73 ± 0.22.

Dans un deuxième temps, nous avons voulu étudier plus en détail l'influence du type de pré-entraînement choisi, en relation avec le nombre de couches cachées $\in \{1,2,3\}$ et le nombre d'unités dans chacune des couches cachées, soit $\in \{300,500,1000,2000\}$, sur la performance de généralisation. Les types de pré-entraînement consistent en : une initialisation aléatoire, un pré-entraînement avec auto-encodeurs de base (AE), un pré-entraînement avec auto-encodeurs débruiteurs (DAE). Pour ce faire, nous avons effectué nombreuses expérimentations sur le jeu de données *rot-bg-img* et avons obtenu les résultats très parlants de la figure 4.2. La première observation se situe au niveau du pré-entraînement. On voit qu'un pré-entraînement, qu'il soit dû à l'usage de AE ou de DAE, permet l'obtention d'une meilleure performance de généralisation qu'une simple initialisation aléatoire des paramètres de l'architecture. De plus, SDAE donne de meilleures performances que son voisin, le SAE. Par ailleurs, l'augmentation du nombre de couches cachées dans un modèle sans pré-entraînement rend l'architecture difficile à entraîner, nuisant à la performance de généralisation. Ainsi, dans le cas de *rot-bg-img*, on passe d'une erreur $\in [60,65]\%$ pour une couche cachée, à une erreur de plus de 85% pour trois couches cachées. Tandis qu'une architecture avec pré-entraînement est capable de tirer un avantage de l'augmentation du nombre de couches cachées. Ainsi l'erreur de classification de test sur *rot-bg-img* diminue d'environ 5% lors du passage d'une couche cachée à trois couches cachées, à 1000 unités chacune. Le tableau 4.4 compare les per-

formances du SAE et du SDAE selon leur nombre de couches cachées, couches dont le nombre d'unités a été fixé à 1000, sur les jeux de données *basic*, *bg-rand* et *rot-bg-img*. On remarque que SDAE bénéficie davantage de l'augmentation du nombre de couches cachées, et obtient dès lors la meilleure performance avec trois couches cachées sur tous les problèmes présentés. Finalement, de retour à la figure 4.2, on remarque que l'augmentation du nombre d'unités par couche cachée paraît plus profitable dans le cas du SDAE où l'on obtiendra une diminution d'environ 10% de l'erreur de généralisation sur une architecture à trois couches cachées du moment que l'on passe de 300 à 1000 unités par couche.

Tableau 4.3 – **Comparaison de SDAE-3 avec d'autres modèles.** Erreurs de test obtenues pour chacun des modèles sur différents problèmes de classification étudiés, accompagnées d'un intervalle de confiance à 95%. Pour chaque ensemble de données, le meilleur résultat est en gras, de même que ceux dont l'intervalle de confiance chevauche celui du meilleur résultat. La valeur de ν est le niveau de bruit qui a été retenu lors de la sélection du modèle, soit le pourcentage de corruption des composantes d'entrées (fmi) pour un bruit MN, sauf dans le cas de *tzan-MPC* où il s'agit de l'écart-type du bruit additif gaussien. À noter que SAE-3 est équivalent à SDAE-3 avec $\nu = 0$. On remarque que SDAE s'avère être de performance égale ou supérieure, dans le cas de *rot*, *rot-bg-img* et *rect-img*, à la performance des autres modèles, sauf pour le cas de *bg-rand* où le modèle DBN-3 l'emporte avec une erreur de test de 6.73 ± 0.22. De plus, sur tous les ensembles de données SDAE obtient de meilleurs résultats que son prédécesseur SAE (sauf pour *convex*, mais la différence n'est pas significative).

Données	**SVM**$_{rbf}$	**SVM**$_{poly}$	**DBN-1**	**SAE-3**	**DBN-3**	**SDAE-3** (ν)
MNIST	**1.40±0.23**	2.00±0.27	**1.21±0.21**	**1.40±0.23**	**1.24±0.22**	**1.28±0.22** (25%)
basic	**3.03±0.15**	3.69±0.17	3.94±0.17	3.46±0.16	3.11±0.15	**2.84±0.15** (10%)
rot	11.11±0.28	15.42±0.32	14.69±0.31	10.30±0.27	10.30±0.27	**9.53±0.26** (25%)
bg-rand	14.58±0.31	16.62±0.33	9.80±0.26	11.28±0.28	**6.73±0.22**	10.30±0.27 (40%)
bg-img	22.61±0.37	24.01±0.37	**16.15±0.32**	23.00±0.37	**16.31±0.32**	**16.68±0.33** (25%)
rot-bg-img	55.18±0.44	56.41±0.43	52.21±0.44	51.93±0.44	47.39±0.44	**43.76±0.43** (25%)
rect	**2.15±0.13**	**2.15±0.13**	4.71±0.19	2.41±0.13	2.60±0.14	**1.99±0.12** (10%)
rect-img	24.04±0.37	24.05±0.37	23.69±0.37	24.05±0.37	22.50±0.37	**21.59±0.36** (25%)
convex	19.13±0.34	19.82±0.35	19.92±0.35	**18.41±0.34**	**18.63±0.34**	**19.06±0.34** (10%)
tzan-MPC	**14.41±2.18**	**15.03±2.21**	18.15±1.43	**16.94±1.95**	18.21±0.85	**16.68±1.30(0.05)**

Ces résultats viennent compléter ceux obtenus par [30] sur les jeux *basic* et *rot* où l'usage du pré-entraînement avec auto-encodeurs de base était validée, où l'augmentation de la profondeur de l'architecture avantageait le réseau avec pré-entraînement et désavantageait celui avec initialisation aléatoire et où l'augmentation du nombre de neurones par couche cachée du SAE, jusqu'à l'atteinte d'un nombre optimal, venait améliorer sa performance. Ici, nous avons reproduit ces expériences sur le jeu de données *rot-bg-img*, en plus d'ajouter de l'information quant au SDAE, à savoir qu'avec un nombre petit d'unités par couche cachée, soit 300, sa performance est semblable à celle d'un SAE, mais que rapidement, l'architecture profite de l'augmentation du nombre d'unités par couche et obtient au final une performance bien au-delà du SAE.

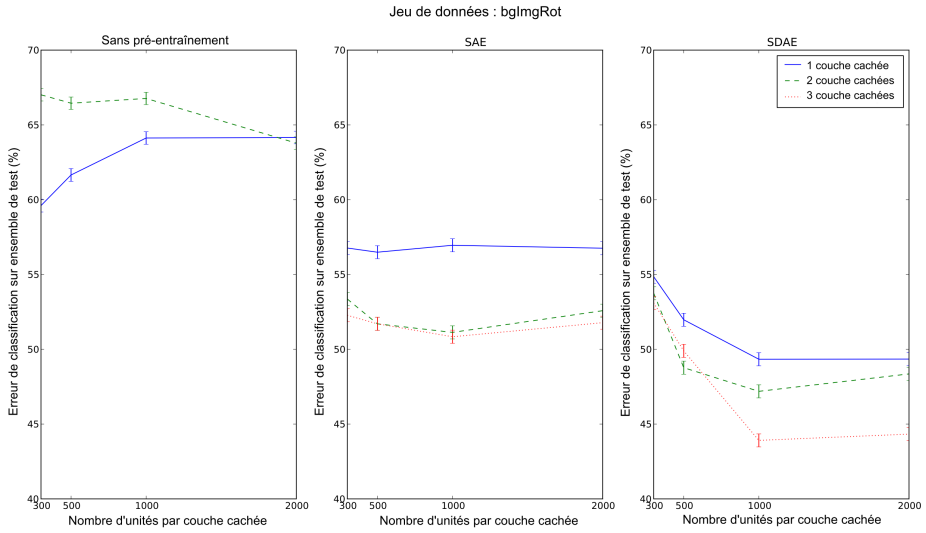


Figure 4.2 – **Performance de classification sur** *rot-bg-img* **selon le type de pré-entraînement, le nombre de couches cachées et le nombre d'unités par couche.** Les architectures avec pré-entraînement démontrent clairement une performance supérieure et tirent profit de l'augmentation du nombre de couches, contrairement à l'architecture avec initialisation aléatoire où l'augmentation du nombre de couches vient rendre l'entraînement du réseau plus ardu. À noter que l'erreur d'un réseau à 3 couches sans pré-entraînement (qui devrait se trouver dans la figure de gauche) est de l'ordre de 89%, donc trop élevée pour être dans la figure. SDAE se démarque de SAE par une performance systématiquement meilleure, du moment que l'on utilise un nombre d'unités cachées de 500 et plus.

4.5 Exploration de SDAE et de la variante avec emphase

Nous allons maintenant nous concentrer sur l'architecture SDAE et en explorer diverses caractéristiques. Nous verrons dans un premier temps l'effet du niveau de bruit imposé lors du pré-entraînement sur la performance de généralisation, dans un second temps l'effet de l'utilisation de différents types de bruit, de même que d'une emphase mise sur la reconstruction des données corrompues, puis nous allons nous interroger sur l'effet d'une corruption des données seulement au niveau de certaines couches de l'architecture plutôt que sur chacune d'entre elles.

4.5.1 Sensibilité au niveau de bruit

Une première question consiste à vérifier comment se comporte la performance en fonction du niveau de bruit. Pour ce faire, nous avons entraîné et testé des architectures à une, deux et trois couches cachées, pré-entraînées sans bruit, ou avec un certain niveau de corruption, sur le jeu de données *rot-bg-img*. La figure 4.3 permet de visualiser dans un premier temps l'avantage de l'usage de bruit lors du pré-entraînement versus l'absence

Tableau 4.4 – **Effet du nombre de couches cachées du SAE et SDAE_mn.** Erreurs de test obtenues sur différents problèmes de classification, selon le type de pré-entraînement et le nombre de couches cachées, reportées avec un intervalle de confiance à 95%. Pour chaque type de pré-entraînement, la meilleure performance est en gras, de même que celles dont l'intervalle de confiance chevauche celle de la meilleure. La fraction de composantes bruitées pour le cas du SDAE est indiquée par ν. Le nombre d'unités par couche est de 1000. Nous observons que l'usage de plus d'une couche cachée augmente la performance de classification. De plus, la combinaison la plus performante est l'usage du pré-entraînement de type SDAE sur une architecture à trois couches cachées.

Variantes	**basic**	**rot**	**bg-Rand**
SAE-1	4.40±0.18	12.16±0.29	18.74±0.34
SAE-2	**3.34±0.16**	**10.25±0.27**	**13.42±0.30**
SAE-3	**3.18±0.15**	**10.30±0.27**	**13.30±0.30**
SDAE-1(ν)	3.34±0.16(10%)	12.78±0.29(25%)	13.49±0.30(40%)
SDAE-2(ν)	**3.12±0.15(25%)**	**10.39±0.27(40%)**	12.67±0.29(10%)
SDAE-3(ν)	**2.84±0.15(10%)**	**10.18±0.27(25%)**	**11.91±0.28(25%)**

de ce dernier, puis de constater qu'il existe un intervalle assez large du niveau de corruption, soit d'environ 20%, qui admet une amélioration considérable de la performance, et ce, nonobstant le nombre de couches cachées de l'architecture. Ainsi, d'un point de vue pratique, il est rassurant de savoir qu'il n'est pas nécessaire pour un expérimentateur d'ajuster très finement le niveau de bruit afin d'obtenir une amélioration satisfaisante.

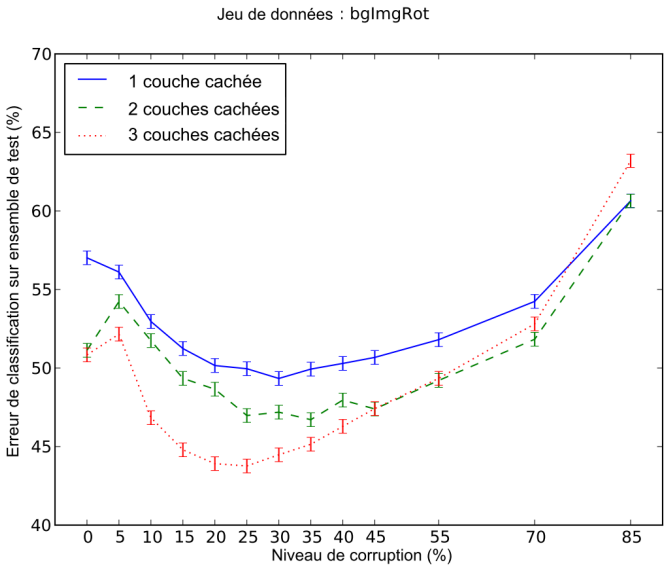


Figure 4.3 – **Effet du niveau de corruption d'un SDAE sur** *rot-bg-img***.** Erreur sur l'ensemble de test accompagnée d'un intervalle de confiance à 95%, en fonction du nombre de couches cachées de (1000 unitées chacune) et du niveau de corruption appliqué (bruit MN) lors du pré-entraînement couche après couche On observe qu'il existe un intervalle d'environ 20% du niveau de corruption qui donne lieu à une amélioration importante de la performance de généralisation, et ce peu importe le nombre de couches cachées que possède l'architecture.

4.5.2 Influence du type de bruit et d'une emphase sur les composantes corrompues

Jusqu'à présent, seul le bruit *masque à zéro* (MN) avait été employé avec l'architecture SDAE. Nous allons maintenant vérifier expérimentalement, sur les problèmes *basic*, *rot* et *bg-rand*, l'effet qu'ont les autres types de bruits envisagés ici, à savoir gaussien (GS) et poivre et sel (PS), sur la performance de classification. Nous allons également

nous pencher sur la variante de l'algorithme DAE qui met une emphase sur la reconstruction des composantes corrompues. Étant donné que le bruit GS s'applique à toutes les composantes de l'entrée, contrairement aux bruits MN et PS qui n'affectent qu'un certain pourcentage des dimensions, la variante de DAE avec emphase n'a de sens que pour les bruits MS et PS. Toutes les variantes de bruit et d'algorithme ont été testées pour une architecture à trois couches cachées avec un nombre optimal d'unités par couche cachée choisi en fonction de l'erreur de validation.

4.5.2.1 Influence sur la performance de la tâche supervisée

La table 4.5 présente les taux d'erreurs obtenus pour chacune des variantes. La comparaison entre les types de bruits, sans emphase, montre que PS ou GS permet à SDAE d'être le modèle le plus performant sur deux des trois ensembles de données étudiés, soit *basic* et *rot*, alors qu'avec le simple masque à zéro, SDAE était au mieux aussi performant que SVM_{rbf} (*basic*), ou aussi performant que SAE-3 et DBN-3 (*rot*). Par contre, du côté du problème *bg-rand*, l'usage des nouveaux bruits sans emphase n'améliore pas les performances. Toutefois, du moment que l'on impose une emphase sur la reconstruction des données corrompues, on remarque une hausse significative des performances, et cette hausse est d'autant plus importante pour les problèmes plus complexes que sont *rot* et *bg-rand*. Cette diminution de l'erreur de généralisation est par ailleurs plus frappante pour le cas de *bg-rand*, faisant passer l'erreur de $10.03 \pm 0.26\%$ à $8.52 \pm 0.24\%$ avec l'usage du bruit PS, performance se rapprochant de la meilleure obtenue jusqu'à présent par DBN-3, soit $6.73 \pm 0.22\%$. En somme, une emphase imposée sur des données bruitées avec PS donne la meilleure performance de tous les modèles pour deux des trois problèmes et la meilleure performance des SDAE pour le cas de *bg-rand*.

4.5.2.2 Influence sur les filtres appris

Observons maintenant les filtres obtenus sur le jeu de données *basic* après un pré-entraînement du premier auto-encodeur débruiteur. Un filtre est le vecteur de poids qu'un neurone de la couche cachée utilise pour calculer son activation (c.a.d. le produit scalaire

de ce vecteur de poids avec l'entrée). La figure 4.4 présente une partie des filtres appris présentés sous forme d'image. Chaque ensemble de filtres est fonction du type et du niveau de bruit utilisés. Pour chaque type de bruit nous avons considéré les réseaux à 1000 neurones par couche et choisi les autres hyper-paramètres (taux d'apprentissage de la phase non-supervisée (*lRate*), nombre d'époques de pré-entraînement (*nEpoq*) et niveau de bruit (ν ou *fmi*) donnant la meilleure erreur sur l'ensemble de validation, puis en gardant fixe les autres hyper-paramètres, nous avons fait varier le niveau de bruit.

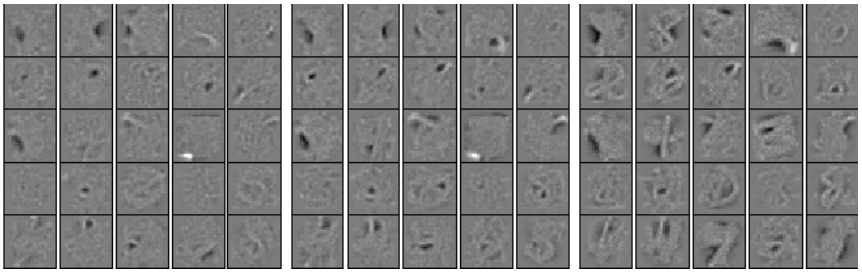
Tableau 4.5 – **Influence du type de bruit d'un SDAE-3 et effet d'une emphase sur la reconstruction des composantes corrompues.** Erreurs de test obtenues avec un SDAE à trois couches cachées sur les problèmes *basic*, *rot* et *bg-rand*. Les résultats sont accompagnés d'un intervalle de confiance à 95%. Pour chaque ensemble de données, le meilleur résultat est en gras, de même que ceux dont l'intervalle de confiance chevauche celui du meilleur résultat. Trois types de bruit ont été expérimentés : masque à zéro (MN), poivre et sel (PS) et bruit gaussien (GS). La valeur de l'hyper-paramètre ν consiste à la fraction des composantes d'entrées corrompues, ou à l'écart-type du bruit GS. Seulement deux niveaux d'emphase sur la reconstruction des composantes corrompues ont été vérifiés : une emphase *moyenne* et *grosse*. Dans les deux cas, le poids sur la reconstruction des dimensions corrompues est de 1, tandis que celui sur les dimensions non corrompues est de 0.5 ou 0 respectivement. En ce qui concerne le bruit gaussien, comme ce dernier est appliqué sur *chacune des dimensions*, la notion d'emphase ne s'applique pas. On observe que sans emphase, le bruit PS et GS améliore les résultats du SDAE sur *basic* et *rot*, le rendant plus performant que ses compétiteurs alors qu'avec le bruit MN, il n'était qu'au mieux aussi performant que SVM_{rbf} (*basic*), ou aussi performant que SAE-3 et DBN-3 (*rot*). Par ailleurs, SDAE-3_{PS} avec emphase, permet une nette amélioration, et cette dernière est d'autant plus intéressante au niveau de *bg-rand*, rapprochant la performance à celle obtenue par DBN-3, soit 6.73±0.22.

Model	**basic**	**rot**	**bg-Rand**
SVM$_{rbf}$	3.03±0.15	11.11±0.28	14.58±0.31
SAE-3	3.46±0.16	10.30±0.27	11.28±0.28
DBN-3	3.11±0.15	10.30±0.27	**6.73±0.22**
SDAE-3$_{MN}(\nu)$	2.98±0.15(25%)	9.53±0.26(25%)	10.30±0.27(40%)
SDAE-3$_{MN}(\nu)$ **+ emph**	**2.76±0.14(25%)**	10.36±0.27(25%)	9.69±0.26(40%)
SDAE-3$_{PS}(\nu)$	**2.66±0.14(25%)**	9.33±0.25(25%)	10.03±0.26(25%)
SDAE-3$_{PS}(\nu)$ **+ emph**	**2.48±0.14(25%)**	**8.76±0.29(25%)**	8.52±0.24(10%)
SDAE-3$_{GS}(\nu)$	**2.61±0.14(0.1)**	**8.86±0.28(0.3)**	11.73±0.28(0.1)

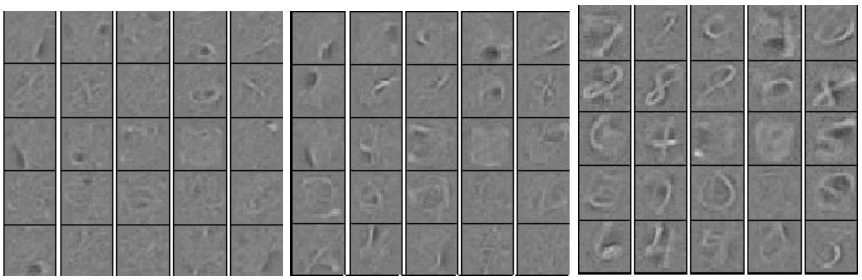
Finalement, chacune des images correspond aux poids issus d'un neurone de la couche cachée. D'abord et avant tout, on constate que pour les trois cas, l'augmentation du niveau de bruit rend certains filtres de moins en moins locaux et augmente le nombre de filtres sensibles à des patrons particuliers. L'usage du bruit MN mène à des filtres s'activant principalement pour des patrons retrouvés en arrière plan des chiffres de *basic*, de forme principalement sphérique ou ovale, tandis que les filtres obtenus avec les bruits PS et GS sont plutôt allongés, semblant capter des structures de plus haute dimension associées aux chiffres eux-mêmes, soit des traits de crayons ou le chiffre en entier lorsque le niveau de bruit est élevé. Il est attendu que plus on élève le niveau de bruit, plus grande sera la perte d'information et ainsi on se retrouve avec des filtres moins locaux, correspondant à un champ récepteur plus large, ceci afin de combler la perte d'information en utilisant des dépendances entre un plus grand nombre de composantes. Ici, les bruits PS et GS corrompent davantage l'image que le bruit MN étant donné que non seulement les composantes associées aux chiffres seront bruitées, mais aussi celles associées à l'arrière-plan. Il est donc normal de voir apparaître des filtres s'apparentant de plus en plus à des chiffres. L'obtention de ce dernier type de filtre associé au niveau le plus élevé de bruit n'est toutefois pas l'objectif à atteindre pour une meilleure performance de généralisation. En effet, on voudrait plutôt observer, plus l'on monte dans l'architecture, des filtres sensibles à des structures de plus en plus abstraites de l'entrée. Par ailleurs, tel que l'a démontré le tableau 4.5, les niveaux à 10%, 25% ou 0.1 d'écart-type sont ceux offrant généralement les meilleures performances. Ainsi, la particularité du bruit GS et PS d'obtenir de filtres sensibles à des structures semblables aux traits de crayon pourrait expliquer l'amélioration de la performance sur la tâche supervisée.

Maintenant, examinons de plus près les filtres appris dans le cas particulier donnant lieu à une nette amélioration de la performance, soit SDAE-3_{ps} avec emphase sur la reconstruction des données corrompues. La figure 4.5 permet de comparer les filtres obtenus sur *basic* avec et sans emphase. À noter que pour cette figure, contrairement à la précédente, on a fixé les hyper-paramètres à leur valeur ayant donné de la meilleure erreur de généralisation sur le problème *basic* lorsqu'une emphase *moyenne* et 10% de bruit PS était appliqué (1000 neurones par couche cachée, un taux d'apprentissage à

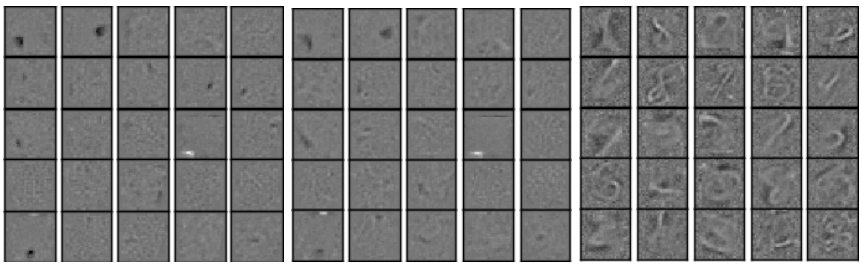
0.05 et 50 époques de pré-entraînement). La qualité des filtres obtenus avec absence ou présence d'emphase est semblable. On observe toutefois que l'utilisation d'une grosse



(a) SDAE_mn (10%, 25%, 50% corruptions)



(b) SDAE_ps (10%, 25%, 50% corruptions et lRate = 0.005)



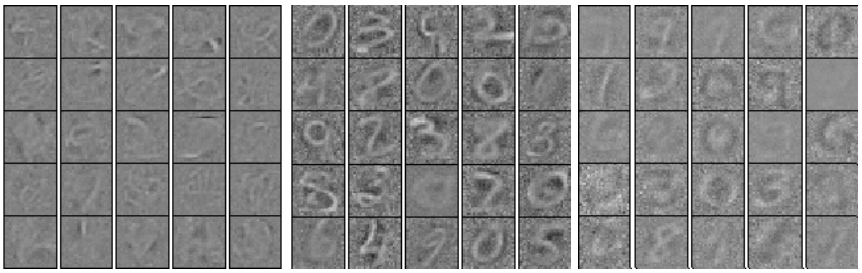
(c) SDAE_gs (0.05, 0.1, 0.3 d'écart-type)

Figure 4.4 – **Filtres appris sur le jeu de données *basic* après pré-entraînement d'un SDAE-3, selon différents types de bruit : masque à zéro (MN), poivre et sel (PS) et gaussien (GS).** Pour chaque type de bruit, trois groupes de filtres sont présentés, chacun étant le fruit d'un entraînement avec une certaine fraction de corruption appliquée à l'entrée, ou avec un certain écart-type pour le cas du bruit GS. Dans tous les cas, l'augmentation du niveau de bruit mène à des filtres de moins en moins locaux. Le cas du bruit MN montre des filtres principalement sensibles à des régions pleines ayant par exemple la forme de gouttes, tandis que les bruits PS et GS induisent des filtres plutôt allongés, reconnaissant par exemple des traits de crayons, voire un chiffre presqu'entier dans le cas du niveau supérieur de bruit.

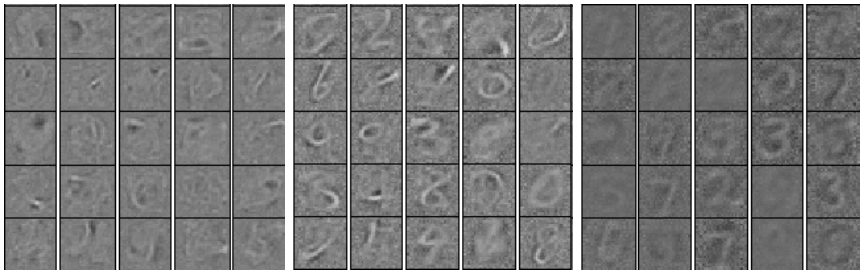
emphase permet, dans le cas d'un niveau de bruit à 10%, l'apparition de filtres plus intéressants, s'activant pour certains traits de crayon, tandis que pour une corruption à 25%, un plus grand nombre de filtres viendront réagir pour des chiffres presqu'entiers. Malgré cela, il reste difficile de comprendre, via l'observation qualitative de ces filtres seuls, la raison de la hausse de performance sur la tâche supervisée, avec utilisation d'une emphase *moyenne*.

4.5.3 Influence du nombre de couches corrompues

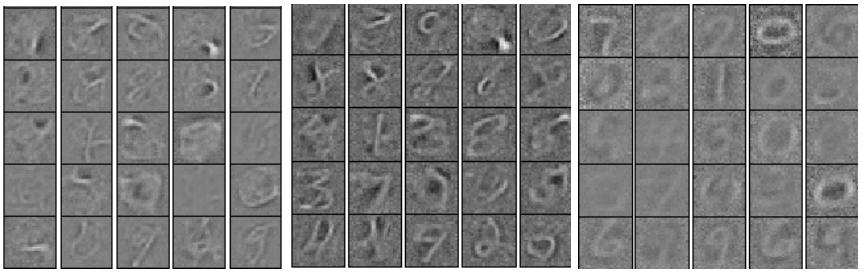
Jusqu'à présent, nous avions toujours appliqué le bruit sur *chacune* des couches d'entrées des différents DAE composant le SDAE. Nous allons maintenant observer l'effet d'une corruption située non plus sur toutes ces couches, mais seulement au niveau du premier auto-encodeur débruiteur, puis au niveau de l'entrée des deux premiers auto-encodeurs et ainsi de suite (en utilisant le même niveau de bruit pour chaque couche). L'idée de base étant d'appuyer la nécessité de pré-entraîner au débruitage non seulement la couche d'entrée, mais aussi les couches subséquentes, ce qui par ailleurs vient appuyer davantage la différenciation entre l'approche proposée par [45] et celle impliquant un simple pré-traitement de bruitage des données d'entrées. Les expériences ont été produites sur les jeux de données *MNIST*, *basic*, *rot* et *bg-rand*. L'architecture utilisée est le SDAE à trois couches cachées composées de 1000 neurones chacune. Les résultats se situent à la table 4.6. Il est à observer d'abord que les bruits PS et GS offrent de meilleures performances sur *MNIST*que le bruit MN jusqu'ici testé. Par ailleurs, on obtient un résultat fort intéressant avec GS appliqué au niveau des deux premières couches, soit $1.06 \pm 0.20\%$. Par ailleurs, la combinaison d'hyper-paramètres donnant la seconde meilleure erreur sur l'ensemble de validation dans le cas GS appliqué au niveau des trois couches cachées, admet une erreur de test semblable, soit $1.04 \pm 0.20\%$. Ensuite, il ressort de ces résultats que dans tous les cas, sauf pour MN sur *rot*, il est avantageux d'appliquer l'apprentissage par débruitage au niveau de plus d'une couche.



(a) SDAE_ps sans emphase (10%, 25%, 50% corruptions, lRate=0.05)



(b) SDAE_ps moyenne emphase (10%, 25%, 50% corruptions)



(c) SDAE_ps grosse emphase (10%, 25%, 50% corruptions)

Figure 4.5 – **Filtres appris sur le jeu de données** *basic* **après pré-entraînement d'un SDAE-3$_{ps}$, sans emphase (4.5(a)), avec *moyenne* (4.5(b)) et *grosse* (4.5(c)) emphase.** Dans les trois cas, les hyper-paramètres permettant l'obtention de la meilleure erreur de généralisation sur *basic*, c'est-à-dire avec emphase *moyenne* et 10% de corruption, ont été utilisés (1000 neurones par couche cachée, un taux d'apprentissage à 0.05 et 50 époques de pré-entraînement). Les filtres sont regroupés sous trois niveaux de corruption appliquée à l'entrée. La différence qualitative des filtres est subtile. Dans le cas d'un niveau de 10%, on remarque toutefois que l'usage d'une emphase et l'augmentation de celle-ci vient faire naître des filtres plus intéressants. Tandis que pour le cas de 25% de corruption, il apparaît qu'avec l'usage d'une *grosse* emphase, un plus quand nombre de filtres s'activent pour des chiffres presqu'entier.

4.5.4 En résumé

Nous avons d'abord montré expérimentalement que le gain en performance obtenu avec un pré-entraînement par auto-encodeur débruiteur ne nécessitait pas une sélection très fine du niveau de bruit. En effet on a pu observer une nette amélioration pour un

Tableau 4.6 – **SDAE-3 - Effet du nombre de couches bruitées selon le type de bruit.** Erreur de généralisation sur les trois problèmes de classification accompagnés d'un intervalle de confiance à 95% en fonction du nombre de couches bruitées. L'architecture utilisée comporte trois couches cachées à 1000 unités chacune. Pour chaque type de bruit, la meilleure performance est en gras, de même que celles avec lesquelles l'intervalle de confiance se chevauche. Les types de bruits considérés sont le bruit *masque à zéro* (MN), *poivre et sel* (PS) et *gaussien* (GS). La fraction de corruption ou l'écart-type du bruit gaussien est indiqué sous la variable v. Ici, nous vérifions l'effet de l'initialisation d'une architecture profonde à l'aide de l'empilement d'un seul DAE suivie de deux AE, versus l'empilement de deux DAE et un seul AE, ou finalement, ce qui avait toujours été utilisé jusqu'à présent, un empilement de trois DAE. Dans un premier temps, on remarque que pour *MNIST*, l'usage du bruit PS ou GS avec SDAE à trois couches bruitées offre de meilleurs résultats qu'avec le bruit MN. Par ailleurs, l'utilisation de deux couches bruités avec GS donne un résultat particulièrement intéressant, soit 1.06±0.20% d'erreur sur l'ensemble de test. Aussi, on observe que dans tous les cas sauf *rot* avec bruit MN et *bg-rand* avec GS, le débruitage effectué sur les trois couches apparaît comme donnant la meilleure performance ou donnant une performance équivalente à celle obtenue lorsque le débruitage s'effectue au niveau des deux premières couches. L'utilisation du débruitage seulement au niveau de la première couche ne semble donc généralement pas aussi efficace que lorsqu'on en fait usage au niveau de deux ou de trois couches cachées.

Variantes SDAE	MNIST	basic	rot	bg-Rand
MN_1(v)	**1.18±0.21(25%)**	**3.08±0.15(25%)**	**9.73±0.26(25%)**	12.96±0.29(25%)
MN_2(v)	**1.10±0.20(10%)**	**2.95±0.15(10%)**	10.34±0.27(40%)	**12.33±0.29(25%)**
MN_3(v)	**1.28±0.22(25%)**	**2.84±0.15(10%)**	**10.18±0.27(25%)**	**11.91±0.28(25%)**
PS_1(v)	**1.22±0.22(10%)**	3.02±0.15(10%)	9.64±0.26(10%)	11.51±0.28(40%)
PS_2(v)	**1.19±0.21(25%)**	2.99±0.15(25%)	**8.84±0.25(25%)**	10.87±0.27(25%)
PS_3(v)	**1.19±0.21(25%)**	**2.66±0.14(25%)**	**9.33±0.25(25%)**	**10.03±0.26(25%)**
GS_1(v)	**1.26±0.22(0.5)**	3.63±0.16(0.05)	10.33±0.27(0.05)	**12.08±0.29(0.5)**
GS_2(v)	**1.06±0.20(0.1)**	**2.71±0.14(0.5)**	**9.06±0.25(0.5)**	**11.80±0.28(0.5)**
GS_3(v)	**1.12±0.21(0.1)**	**2.56±0.14(0.15)**	**8.71±0.25(0.5)**	12.49±0.29(0.1)

intervalle relativement large de niveau de bruit. Aussi, bien que le choix du type de bruit a clairement une influence, nous avons observé des améliorations par rapport au pré-entraînement avec auto-encodeurs ordinaires pour trois types de bruits très différents. Nous avons par ailleurs montré qu'un pré-entraînement par débruitage effectué sur au moins deux des trois couches était plus avantageux que lorsque seulement employé pour l'entraînement de la première couche. Aussi la variante d'auto-encodeur débruiteur avec emphase imposée sur la reconstruction des composantes bruitées, pour un bruit *poivre et sel* semble donner lieu aux meilleures performances.

Explorons maintenant la qualité des filtres de premier niveau appris par un DAE sur des parties d'images naturelles et comparons-les avec ceux appris dans le même contexte par un AE ordinaire.

4.6 Comparaison qualitative des filtres appris sur des parties d'images naturelles

Outre les filtres appris sur *basic*, nous avons étudié ceux appris sur des parties d'images naturelles, soit *olsh-12x12*, selon différents facteurs. Entre autres, nous avons voulu vérifier si un AE et un DAE entraînés sur *olsh-12x12*, étaient en mesure d'apprendre, tout comme le modèle parcimonieux de [36] un type de filtre semblable aux Gabor, soit le type de filtre dont se sert la zone V1 du cortex visuel afin d'extraire les contours d'une image. Les facteurs étudiés sont : le type du bruit, le niveau de bruit, le type de représentation, l'usage de matrices d'encodage et de décodage liées ou non-liées, de même que l'emphase mise sur la reconstruction des composantes corrompues. Par type de représentation, on entend : sous-complète, complète et sur-complète, où la taille de la représentation est plus petite, égale et plus grande que la taille de l'entrée, respectivement. L'hypothèse première formulée est qu'un auto-encodeur ordinaire apprenant une représentation complète ou sur-complète, avec des matrices de connexions non-liées, aura tendance à apprendre une solution triviale. Il suffit que la composition de l'encodeur et du décodeur soit l'identité ; par exemple, il est possible de rester dans le régime linéaire de la sigmoïde en apprenant des poids d'encodage très petits et des poids de décodage très grands afin d'être en mesure de reproduire telle quelle l'entrée. On s'attendra dès

lors à ce que les filtres appris ne soient pas très intéressants. Tandis qu'un auto-encodeur débruiteur, quelle que soit la taille de sa représentation, parce qu'il doit *débruiter son entrée*, ne peut pas se permettre d'apprendre une solution triviale. Ainsi, on s'attend à ce qu'il apprenne des filtres ayant une structure intéressante possiblement semblables aux filtres de Gabor. Toutefois, le comportement attendu chez l'auto-encodeur ordinaire lorsqu'il est question de matrices de connexions d'encodage et de décodage liées, est moins clair. Il est possible que le fait de lier les matrices, impose une contrainte au modèle, l'empêchant d'apprendre la solution triviale et cela même si la taille de sa couche cachée est égale ou plus grande que l'entrée, et ainsi lui donner l'occasion d'apprendre une représentation intéressante. En ce qui a trait à l'usage d'emphase sur la reconstruction des composantes corrompues, étant donné que nous avons démontré une nette amélioration des performances de classifications sur la majorité de jeux de données lorsqu'il y avait application de cette emphase, il sera intéressant de constater ce qu'il advient de la qualité des filtres dans ce cas particulier.

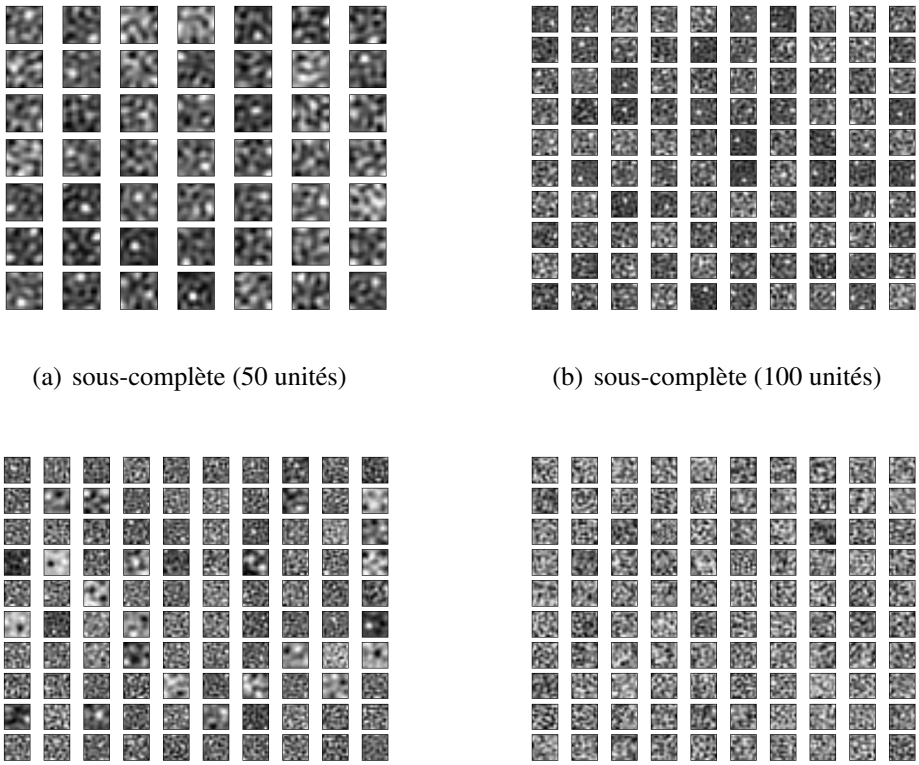
4.6.1 Protocole

Le jeu de données utilisé est celui des parties d'images naturelles, *olsh-12x12*, présenté à la section 4.1. Les trois types de bruits qui ont été testés sont : le masque à zéro (MN), le bruit poivre et sel (PS), de même que le bruit gaussien (GS). Étant donné que les caractéristiques d'entrées se situent dans $[-2, 2]$, nous avons utilisé les valeurs suivantes de poivre et de sel respectivement : $\{-1.5, 1.5\}$. À noter qu'à moins d'être spécifié, les expérimentations suivantes impliquent des matrices d'encodage et de décodage liées.

4.6.2 Influence du bruit et de l'usage de matrices non-liées

D'abord, nous avons voulu explorer les filtres d'un AE ordinaire, soit sans l'usage d'un bruit lors de son entraînement. Tel qu'attendu et montré à la figure 4.6, les filtres ne sont guère qualitativement intéressants, et ce peu importe que l'on utilise une représentation sous-complète qui permet d'éviter que le AE apprenne une solution triviale. Des expérimentations supplémentaires ont permis de confirmer qu'il en était de même pour

le cas de matrices non-liées, (résultats non montrés car redondants).



(a) sous-complète (50 unités)

(b) sous-complète (100 unités)

(c) Complète (144 unités)

(d) Sur-Complète (200 unités)

Figure 4.6 – **Filtres naturels issus d'un AE avec matrices liées, en fonction du type de représentation : sous-complète, complète et sur-complète.** On observe que beaucoup des filtres obtenus avec un auto-encodeur de base (particulièrement dans le cas surcomplet) sont similaires à ce qu'on obtiendrait avec une initialisation au hasard. D'autres semblent être des détecteurs de pixels qui vont simplement répliquer un pixel de l'entrée (ou possiblement moyenner un voisinage très local d'un pixel). Aucune structure vraiment intéressante ne semble avoir été apprise.

Ce n'est qu'avec l'usage d'un bruit, poids liés ou non, que l'on verra finalement émerger des filtres beaucoup plus intéressants, moins locaux, voire même des filtres semblables aux filtres de Gabor obtenus par [36]. L'image 4.7 montre ces filtres, que l'on nommera filtres *à grille* (figure 4.7(b)) et *filtres de Gabor* (figures 4.7(c) et 4.7(d)), ainsi obtenus pour le cas de matrices liées et représentations sous-complètes possédant

100 unités. Les deux premiers types de filtres se sont avérés possibles grâce à l'usage du bruit masque à zéro, et varient selon le pas d'apprentissage utilisé. Tandis que les filtres de Gabor ont été obtenus par l'usage du bruit poivre et sel ou du bruit gaussien. Les filtres semblables aux filtres Gabor, sont ceux sur lesquels nous allons nous attarder, s'agissant de filtres dont la fonction sert à isoler les différents contours de l'image d'entrée, rappelant la fonction des cellules de la région V1 du système visuel du cerveau. La figure 4.8 montre qu'un niveau minimum de bruit gaussien est nécessaire afin de voir apparaître des filtres de Gabor.

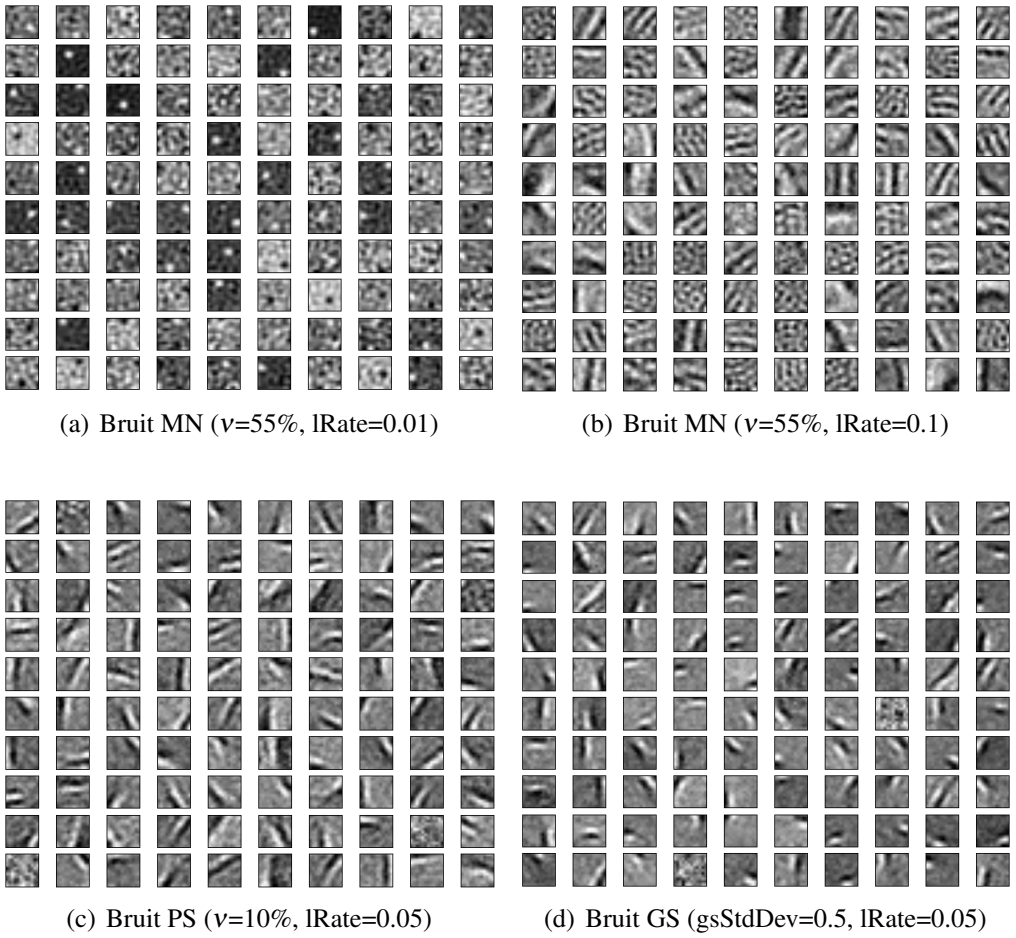
4.6.3 Effet de la taille de représentation

Nous avons observé que pour chacun des types de filtres, Gabor inclus, l'usage de représentations plus petites, complètes ou sur-complètes n'affecte pas le type de filtres obtenus. Qui plus est, dans le cas du bruit gaussien, on constatera que l'augmentation du nombre d'unités cachées vient émettre davantage de filtres Gabor. La figure 4.9 montre qu'effectivement, la taille de la représentation va influencer le rapport de filtres de type à grille versus la quantité de filtres Gabor. Ainsi, plus on augmente la taille de la couche cachée, plus les filtres *à grille* laissent place à des filtres de Gabor. Une explication serait qu'un nombre satisfaisant d'unités cachées permet que chacune d'entre elles s'active pour un trait d'un certain angle et d'une certaine position, tandis qu'un nombre moindre d'unités s'avère insuffisant pour le codage des différentes combinaisons principales d'angles et positions associées à un trait, nécessitant l'usage de filtres s'activant pour un nombre supplémentaire de dimensions, tels les filtres *à grille*.

4.6.4 Effet d'une emphase sur les données corrompues

Voyons maintenant l'effet qualitatif sur les filtres d'une emphase sur la reconstruction des composantes corrompues. Reprenons le cas d'un DAE utilisant du bruit poivre et sel. La figure 4.10 montre les filtres pour le cas de représentations sous-complètes selon qu'il y ait ou non emphase. On constate qu'à première vue, les filtres sont très semblables, quoi qu'on puisse remarquer qu'avec emphase et augmentation de cette dernière, les

filtres Gabor obtenus sont d'épaisseur plus élevée, réagissant donc à une structure de plus haute dimension. Nous avons par ailleurs vérifié si une emphase avec le bruit MN permettait l'obtention de filtres semblables aux filtres de Gabor, mais il n'en fût rien.



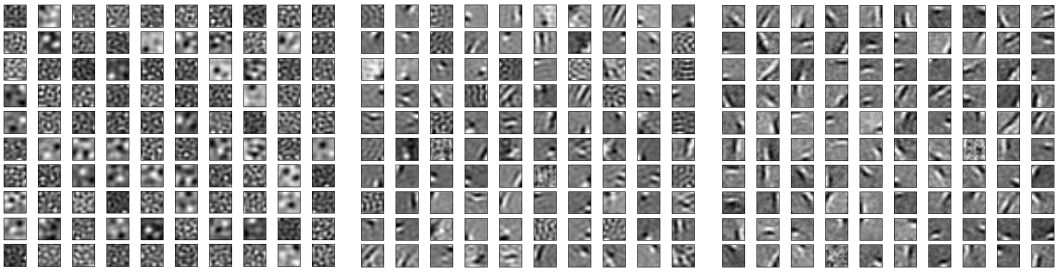
(a) Bruit MN (ν=55%, lRate=0.01) (b) Bruit MN (ν=55%, lRate=0.1)

(c) Bruit PS (ν=10%, lRate=0.05) (d) Bruit GS (gsStdDev=0.5, lRate=0.05)

Figure 4.7 – **Filtres naturels issus d'un DAE avec représentation sous-complète, en fonction du type de bruit utilisé : masque à zéro, poivre et sel et gaussien.** Avec l'usage de bruit, les filtres sont plus spécifiques et moins locaux que lorsqu'il y a absence de bruit durant l'entraînement. On remarque aussi l'importance du choix du taux d'apprentissage dans le cas du MN ; avec un taux d'apprentissage trop petit, (a), les filtres, des détecteurs de pixels, sont locaux, et peu intéressants, tandis qu'avec un taux d'apprentissage plus élevé (b), on voit naître des filtres *à grille*. Aussi, seuls les bruits PS et GS permettent l'obtention de filtres semblables aux filtres bien connus appelés filtre de Gabor (c et d).

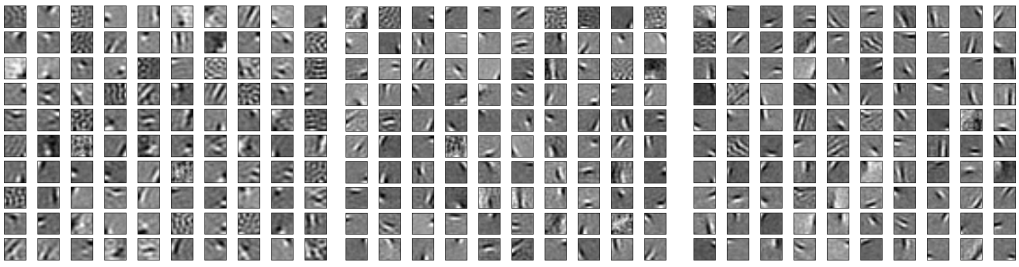
4.6.5 En résumé

À la lumière de ces résultats, il semble que le modèle AE ne puisse éviter d'apprendre une solution triviale et admet donc des filtres peu intéressants, que ses matrices soient ou non-liées. À l'opposé, le DAE obtiendra dans les deux cas des filtres particuliers, et ce nonobstant le type de représentation. Quant à l'emphase sur la reconstruction des entrées corrompues, outre l'augmentation de l'épaisseur des filtres Gabor obtenus, elle ne donne pas lieu à des différences qualitatives importantes sur les filtres appris avec ces données. Voyant la capacité supérieure du DAE d'apprendre des filtres sensibles à une structure



(a) Ecart type = 0.1 (b) Ecart type = 0.3 (c) Ecart type = 0.5

Figure 4.8 – **Visualisation des filtres naturels issus d'un AE avec bruit gaussien pour différentes valeurs d'écart-type.** On remarque que l'augmentation du niveau de bruit fait naître des filtres intéressants, dits filtres de Gabor.



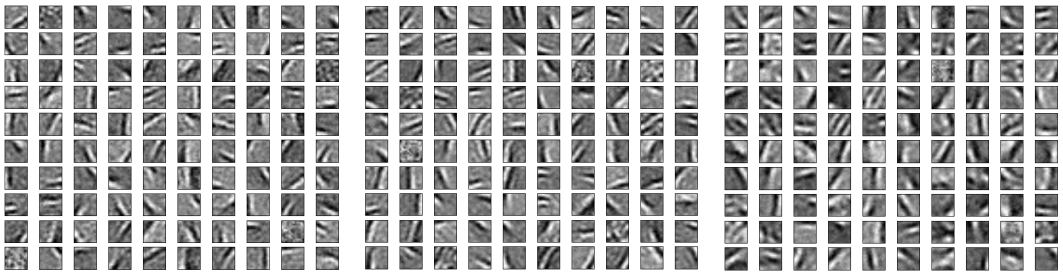
(a) Sous-complète (b) Complète (c) Sur-complète

Figure 4.9 – **Visualisation des filtres naturels issus d'un AE avec bruit gaussien pour différent type de représentation.** On remarque que l'augmentation du nombre d'unités cachées vient diminuer le nombre de filtres dits "à grille" pour augmenter le nombre de filtres de Gabor.

intéressante on comprend mieux qu'il soit en mesure, de par l'extraction de caractéristiques plus adéquates, de donner lieu à une meilleure performance de classification que les AE.

4.7 Démystifier certaines croyances concernant le pré-entraînement par SDAE

L'idée de [45] de faire usage d'un empilement de modules débruiteurs pour le pré-entraînement non-supervisé fait naître certaines présomptions dont deux que nous allons éclairer. La première consiste à savoir si cette méthode diffère d'un simple entraînement sur des données bruitées lors d'un pré-traitement, procédure dont le succès a été démontré par le passé [2, 23]. La seconde fait référence à une possible croyance selon laquelle l'entraînement d'un avec bruit gaussien reviendrait à peu près au même que l'entraînement sans bruit mais avec régularisation L2. Elle trouve son origine dans l'équivalence montrée par [11] entre l'entraînement avec bruit gaussien et la régularisation Tikhonov, mais qui n'est valable que dans la limite d'un niveau de bruit très petit.



(a) Pas d'emph. (α=1,γ=1) (b) Moyenne emph. (α=1,γ=0.5) (c) Grosse emph. (α=1,γ=0)

Figure 4.10 – **Visualisation des filtres naturels issus d'un AE avec représentations sous-complètes et bruit poivre et sel à** 10% **d'entrées corrompues, selon qu'il y ait emphase ou non sur la reconstruction des données corrompues.** On remarque une similarité entre les filtres, outre l'épaisseur des filtres de Gabor qui augmente avec l'utilisation d'une emphase et augmentation de cette dernière.

4.7.1 Différencier l'apprentissage par débruitage de l'apprentissage sur un ensemble bruité

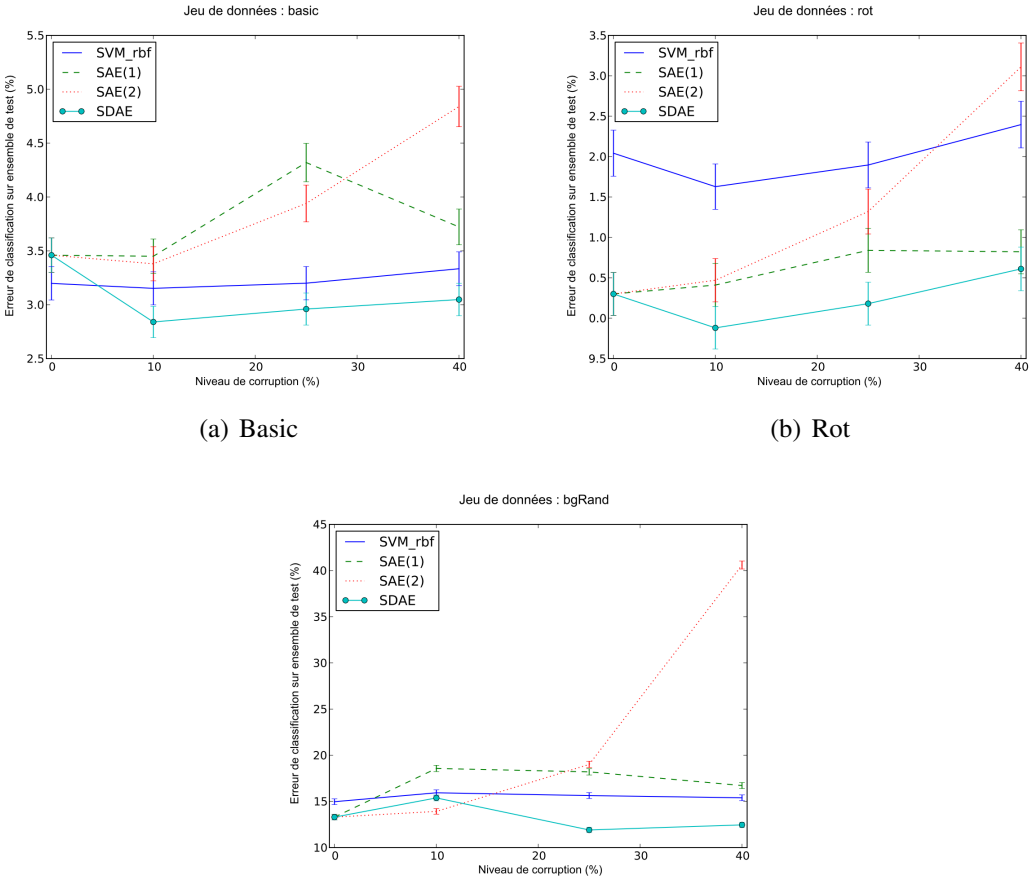
Par le passé, il a été montré que le fait d'entraîner un réseau de neurones sur des données issues d'un pré-traitement impliquant du bruitage, augmentait la performance pour une tâche supervisée [23, 42]. Il est naturel de faire le rapprochement entre cette technique et le pré-entraînement induit par SDAE, car après tout, nous pouvons considérer ce dernier comme étant un entraînement supervisé sur une entrée bruitée où la cible, de haute dimension, s'avère être l'entrée avant bruitage. Cette équivalence suppose donc qu'un SAE entraîné sur un ensemble prétraité de manière à comporter des dimensions bruitées, se verrait obtenir des résultats de généralisation semblable au SDAE. En plus de quoi, on pourrait croire qu'un SVM entraîné sur des données bruitées obtiendrait des résultats surpassant ceux du SDAE, qui jusqu'à présent s'avérait être compétitif avec, voire plus performant dans certains cas que le SVM. Nous avons voulu vérifier expérimentalement s'il existe un avantage au principe d'apprentissage par débruitage proposé par [45] par rapport à la technique d'apprentissage sur des données bruitées suggérée par [23, 42]. Pour ce faire, nous avons vérifié si le SVM à noyau gaussien, puis le SAE-3, chacun entraîné sur des données issues d'un pré-traitement impliquant du bruitage, offraient une performance supérieure ou semblable à celle du SDAE. Les architectures SAE-3 et SDAE-3 employées sont celles à trois couches cachées, chacune de 1000 neurones. Les trois ensembles de données originales utilisées sont *basic*, *rot* puis *bg-rand*, et le pré-traitement consiste à appliquer un bruit sur une certaine fraction des composantes de chacune des entrées. Deux bruits ont été testés : MN et PS. Les résultats en ce qui a trait à l'utilisation du bruit MN se situent à la figure 4.11. À noter que deux variantes ont été testées pour le cas de SAE-3. Pour les deux variantes, le pré-entraînement a été effectué sur des données bruitées. La différence se situe au niveau de l'entraînement supervisé : pour la courbe SAE(1), l'entraînement supervisé a été effectué sur les données originales, tandis que dans le cas SAE(2), des données bruitées ont été utilisées. Ainsi, on y voit que, pour les trois différents jeux de données, l'usage d'un ensemble bruité pour l'entraînement du SVM_{rbf}, tout comme pour celui de SAE-3, que ce soit au niveau du

pré-entraînement **seul** ou au niveau du pré-entraînement **et** de l'entraînement supervisé, ne permet pas d'égaler la performance du SDAE-3, ce dernier obtenant les meilleurs résultats. Nous tirons la même conclusion pour le cas de l'usage du bruit poivre et sel (résultats non montrés car redondants). Ainsi, l'emploi d'un modèle débruiteur (SDAE) n'est pas équivalent à un modèle SAE entraîné sur des données prétraitées de manière à inclure du bruit. Par ailleurs un SVM_{rbf} entraîné sur un ensemble bruité ne rattrape pas la performance du SDAE.

4.7.2 Différencier l'apprentissage SAE avec régularisation L2 et SDAE avec bruit gaussien

Il a été présumé qu'un entraînement avec bruit gaussien serait équivalent à une l'application d'une régularisation Tikhonov [11]. En effet, l'addition d'un bruit gaussien lors d'une tâche de *régression linéaire* revient au même que l'application d'une régularisation L2 de coefficient égale à la variance du bruit précédent. Un parallèle à faire serait qu'un SAE avec application d'une régularisation L2 sur les poids lors de son pré-entraînement, serait équivalent à un SDAE avec usage du bruit gaussien lors de son pré-entraînement. Toutefois étant donné les multiples transformations non-linéaires imposées par un réseau de neurones, une régularisation de Tikhonov ne se réduit plus à une simple régularisation L2. Qui plus est, l'équivalence montrée par [11] est basée sur une expansion de Taylor et n'est donc valable que dans la limite d'un bruit très petit. Nous avons effectué des expériences pour vérifier empiriquement s'il y avait une différence importante entre ce qu'apprenait un SAE avec régularisation L2 et un SDAE avec bruit gaussien. Nous allons pour ceci comparer qualitativement les filtres de la première couche obtenus dans les deux cas sur les données *olsh-12x12*, ainsi que la performance de classification sur le jeu de donnée*rot-bg-img*. Les valeurs testées pour le coefficient de régularisation de même que pour l'écart-type du bruit gaussien sont indiquées dans la table 4.7. La figure 4.12 montre les filtres obtenus après pré-entraînement d'un AE avec régularisation L2, puis ceux obtenus après pré-entraînement d'un DAE avec bruit gaussien. Pour toutes les valeurs d'hyper-paramètres testées, il nous a été impossible d'apprendre des filtres intéressants avec AE_{L2}, tandis que des filtres semblables à des

filtres de Gabor ont pu être appris par DAE$_{GS}$. Finalement, en ce qui a trait à la tâche



(a) Basic

(b) Rot

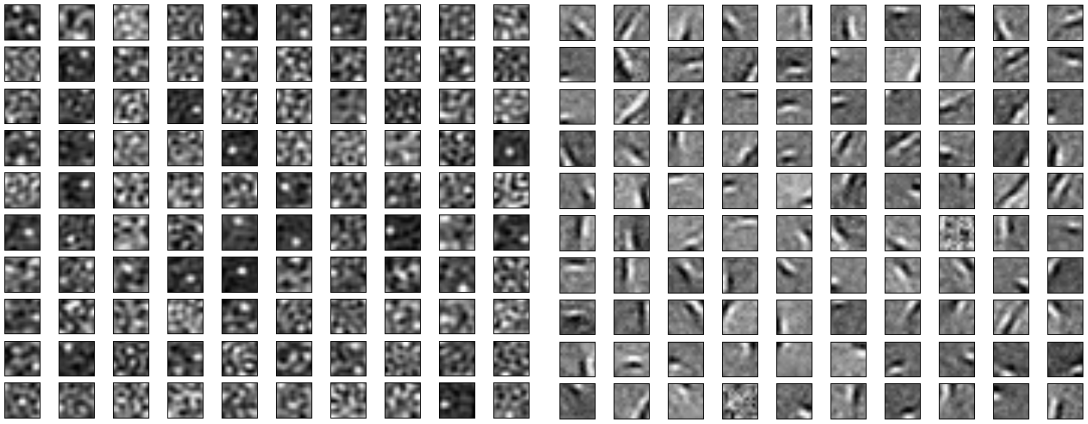
(c) BgRand

Figure 4.11 – **Comparaison de la performance de généralisation d'un SDAE-3, avec les performances du SVM$_{rbf}$ et de SAE-3, tous deux d'abord entraînés sur l'ensemble original d'entraînement, puis sur une version bruitée de ce dernier avec le bruit masque à zéro.** Les performances montrées sont celles obtenues sur trois jeux de données, soit *basic*, *rot* et *bg-rand*, en fonction d'une fraction croissante ν de composantes bruitées. Deux courbes sont montrées pour SAE : la performance lorsque **SAE(1) : seul son pré-entraînement** est effectué sur un ensemble bruité, puis lorsque **SAE(2) :** son pré-entraînement **et** son entraînement supervisé sont effectués sur un ensemble bruité. Pour les réseaux, nous avons utilisé 1000 neurones par couche cachée. La barre d'erreur représente l'intervalle de confiance à 95%. On remarque que pour les trois jeux de données, ni SVM$_{rbf}$, ni SAE-3 entraînés sur des données bruitées n'obtiennent une erreur de test aussi basse que SDAE-3.

de classification sur *rot-bg-img*, la performance de SDAE-3_{GS} s'avère significativement meilleure que celle de SAE-3_*L2*, soit 47.47±0.44% versus 50.76±0.44%.

Tâche d'apprentissage de filtres sur des parties d'images naturelles	
Hyper-paramètre	**Valeurs**
Coefficent L2	{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5, 1.0}
Ecart type du bruit gaussien	{0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0}
Tâche de classification sur *rot-bg-img*	
Hyper-paramètre	**Valeurs**
Coefficent L2	{0.00005, 0.0005, 0.005, 0.05, 0.1, 0.5, 1.0}
Écart type du bruit gaussien	{0.1, 0.5}

Tableau 4.7 – **Valeurs d'hyper-paramètres testées pour la comparaison entre régularisation L2 et entraînement avec bruit gaussien.**



(a) AE_{L2} (b) DAE_{GS}

Figure 4.12 – **Comparaison des filtres naturels obtenus avec AE_{L2} et DAE_{GS}.** Pour un large intervalle de coefficient de régularisation ($\in [0.0001, 1]$) et d'écart-type pour le bruit gaussien ($\in [0.005, 0.1]$) testés, nous avons été en mesure de démontrer la différence au niveau qualitatif des filtres naturels appris d'une part avec une régularisation L2 lors du pré-entraînement (4.12(a)), puis d'une autre part ceux appris avec pré-entraînement avec bruit gaussien (4.12(b)). Les filtres appris par AE_{L2} versus DAE_{GS} sont qualitativement clairement différents. Seul DAE_{GS} est en mesure d'apprendre des filtres semblables à des filtres de Gabor.

4.8 Expérimentation des nouvelles stratégies de pré-entraînement

Suite à la démonstration de l'effet bénéfique au niveau des représentations apprises comme au niveau de la performance de généralisation dû à l'inclusion d'un bruit dans le pré-entraînement d'une architecture profonde, nous avons voulu élargir la question et tenter de débusquer de nouveaux modules de pré-entraînement impliquant du bruit dans le but de potentiellement obtenir des performances supérieures de classification, voire apprendre des filtres qualitativement plus intéressants. Ces variantes consistent en l'auto-encodeur rebruiteur (RAE) utilisé avec bruit MN et PS, l'auto-encodeur comblant les données manquantes de type *binomial missing* (MAE$_{BM}$) et *one if missing* (MAE$_{OM}$), puis la machine de Boltzmann restreinte intégrant un processus de bruitage lors de la phase de propagation de son entraînement (RBM$_{fprop}$) ou lors de la mise à jour de ses paramètres (RBM$_{update}$). Étant donné la différence de performance de classification de même que celle au niveau de la qualité des filtres appris d'une architecture DBN basée sur des RBM ayant une couche visible de type *binaire* versus utilisation d'une couche visible de type *exponentielle tronquée* [7, 29], il est important de rappeler ici que nous avons employé une couche d'entrée de type *exponentielle tronquée*.

4.8.1 Performances de généralisation

En premier lieu, nous comparons SRAE puis les deux versions de SMAE avec SDAE$_{mn}$. Chacune des architectures utilisées possède trois couches cachées, et ont été testées sur les jeux de données *basic*, *rot* puis *bg-rand*. Les résultats obtenus se situent dans la table 4.8. On observe que SRAE avec bruit PS ou MN offre des résultats prometteurs, en améliorant la performance sur les trois jeux de données. Par ailleurs SMAE$_{BM}$ offre la meilleure performance sur *bg-rand*. De plus, il apparaît que sur deux des trois jeux de données, SMAE$_{OM}$ est un peu plus efficace. Quant à l'application d'une emphase sur la reconstruction des composantes manquantes, on ne remarque aucune différence significative.

Finalement, nous avons testé les variantes de l'architecture DBN sur les jeux de données *basic* et *rot*. Tel que décrit à la section 3.1, nous avons voulu vérifier si l'utilisation

d'une version partiellement bruitée de v^0, la couche visible de la phase positive, à deux moments différents du pré-entraînement couche par couche, permettrait l'obtention de meilleurs résultats. À titre de rappel, les variantes étudiées sont DBN$_{fprop}$, soit l'usage de la version bruitée lors du calcul de h^0, et DBN$_{update}$, lors de la mise à jour des paramètres. Les résultats situés dans 4.9 permettent de conclure que l'intégration du bruit n'améliore pas les résultats : si utilisé au niveau de la mise à jour des paramètres, elle offre des performances semblables à l'architecture originale, tandis que l'inclusion du bruit au niveau de la propagation viendra plutôt nuire à la performance.

Tableau 4.8 – **Comparaison des stratégies de pré-entraînement.** Erreurs de test obtenues avec une architecture à trois couches cachées en fonction de différents modules de pré-entraînement appliqués sur les jeux de données *basic*, *rot* et *bg-rand*. Les résultats sont accompagnés d'un intervalle de confiance à 95%. Pour chaque ensemble de données, le meilleur résultat en gras, de même que ceux dont l'intervalle de confiance chevauche celui du meilleur résultat. La valeur de ν consiste au pourcentage d'entrées bruitées ou manquantes dans le cas SMAE. Les variantes étudiées sont l'auto-encodeur rebruiteur (RAE) avec bruit MN et PS, l'auto-encodeur robuste aux données manquantes de type BM (SMAE$_{BM}$) et OM (SMAE$_{OM}$), et finalement à titre de référence, l'architecture à auto-encodeurs ordinaires (SAE) et celle avec auto-encodeurs débruiteurs (SDAE). Dans tous les cas, nous nous sommes restreints à 1000 neurones par couches cachées. On remarque que pour les trois problèmes étudiés, RAE avec bruit MN ou PS améliore les résultats et devient le plus performant sur DAE et AE. Les performances de SMAE$_{BM}$ et SMAE$_{OM}$ ne sont pas loin derrière et SMAE$_{BM}$ s'avère le meilleur modèle à utiliser pour le cas du problème le plus complexe, soit *bg-rand*. Pour ce qui est de l'emphase imposée sur la reconstruction des données manquantes, on ne note aucune amélioration significative de la performance de classification sur les trois jeux de données.

Variantes	**basic**	**rot**	**bg-Rand**
SAE-3	3.18 ± 0.15	10.30 ± 0.27	13.30 ± 0.30
SDAE-3_$mn(\nu)$	**2.84±0.15(10%)**	**10.18±0.27(25%)**	$11.91 \pm 0.28(25\%)$
SRAE-3_$mn(\nu)$	**2.90±0.15(10%)**	**9.97±0.26(10%)**	**11.03±0.27(25%)**
SRAE-3_$ps(\nu)$	**2.65±0.14(10%)**	**9.82±0.26(10%)**	$12.49 \pm 0.29(10\%)$
SMAE-3$_{BM}(\nu)$	$3.45 \pm 0.16(10\%)$	$11.52 \pm 0.28(10\%)$	**10.60±0.28(40%)**
SMAE-3$_{OM}(\nu)$	$3.12 \pm 0.15(10\%)$	**10.11±0.26(25%)**	$12.57 \pm 0.28(10\%)$
SMAE-3$_{BM}(\nu)$ **emph**	$3.28 \pm 0.16(40\%)$	$11.88 \pm 0.28(40\%)$	**10.73±0.27(10%)**
SMAE-3$_{OM}(\nu)$ **emph**	$3.28 \pm 0.16(25\%)$	**9.83±0.26(10%)**	$12.35 \pm 0.29(10\%)$

Tableau 4.9 – **Comparaison des performances de classification de DBN, DBN$_{update}$ et DBN$_{fprop}$** Erreurs de test obtenues avec DBN à trois couches cachées et deux variantes faisant usage d'une version partiellement bruitée de v^0, soit la couche visible de la phase positive, lors du pré-entraînement couche par couche : DBN$_{fprop}$ avec utilisation de la version partiellement bruitée lors de la propagation et DBN$_{update}$ avec utilisation de la version partiellement bruitée lors de la mise à jour des paramètres. Le bruit utilisé est le masque à zéro (MN). Pour chacune des architectures, nous avons restreint nos expériences à 1000 neurones par couche cachée. Les jeux de données testés sont *basic* et *rot*. Les résultats sont accompagnés d'un intervalle de confiance à 95%. Pour chaque ensemble de données, le meilleur résultat en gras, de même que ceux dont l'intervalle de confiance chevauche celui du meilleur résultat. La valeur de v consiste au pourcentage de composantes bruitées de v^0. On remarque que pour les deux jeux de données, l'architecture de base DBN-3 comme la variante DBN-3$_{update}$ admet des résultats semblables, tandis que DBN-3$_{fprop}$ nuit à la performance.

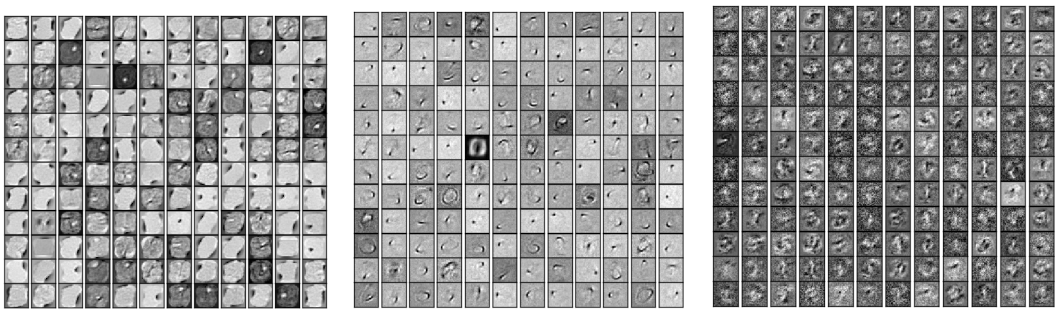
Modèles	**basic**	**rot**
DBN-3	**3.29±0.15**	**10.53±0.27**
DBN-3$_{update}(v)$	**3.77±0.17(40%)**	**10.58±0.27(10%)**
DBN-3$_{fprop}(v)$	4.95±0.20(25%)	14.80±0.31(10%)

4.8.2 Qualité des filtres appris

Afin de nous amener vers une meilleure compréhension de ces résultats, penchons-nous sur les filtres obtenus entre les entrées et la première couche cachée après le pré-entraînement pour le cas de *basic*. Nous avons vu précédemment, via la figure 2.4 que sans l'application d'un bruit, certains filtres semblent n'apprendre aucune caractéristique intéressante, comme s'ils résultaient tout simplement d'une initialisation hasardeuse. Alors que l'apprentissage par débruitage avec un bruit masque à zéro [45] permettait à ces mêmes filtres de devenir des détecteurs de caractéristiques intéressantes. Aussi, l'augmentation de la fraction de composantes d'entrée masquées à zéro rendait les filtres moins locaux et plus sensibles à des structures de plus haute dimension. Vérifions si les nouvelles variantes engendrent des filtres tout aussi, ou davantage intéressants. Il est à noter que dans le cas du modèle SMAE-3, comme les entrées ont été doublées avant de se faire attribuer des poids, seuls les filtres associés aux entrées d'indice pair sont considérés. Ainsi, la figure 4.14 révèle les filtres obtenus avec le jeu de données *basic* selon qu'il

est question de la variante SDAE-3$_{mn}$, SRAE$_{\{mn,ps\}}$ ou SMAE$_{\{BM,OM\}}$. Nous pouvons constater que tous les modèles présentés, sauf SRAE$_{ps}$, apprennent des filtres codant pour des structures plutôt pleines qui caractérisent l'arrière-plan des chiffres tandis que SRAE$_{ps}$ aura appris des filtres réagissant aux coups de crayon. Par ailleurs, on remarquera que SMAE-3 présente des filtres à gouttes particulièrement rondes, tandis que les autres auront des filtres s'activant pour des formes plus allongées, telles que des ovales. Les filtres de SRAE$_{ps}$ permettant la capture des contours de l'image, soit une caractéristique plus abstraite de la distribution d'entrée, pourrait expliquer l'obtention de la meilleure performance de classification sur *basic*, avec $\nu = 0.10\%$.

Voyons maintenant les filtres obtenus sur *basic* avec un RBM et les variantes RBM$_{update}$ et RBM$_{fprop}$. La figure 4.13 permet de constater que RBM$_{update}$ apprend des filtres intéressants, répondant à des structures soit en forme de goutte ou de traits courbés de crayon, tandis que ceux de RBM$_{fprop}$ n'ont pas de caractéristique clairement identifiable.



(a) RBM traditionel (b) RBM$_{update}$ (c) RBM$_{fprop}$

Figure 4.13 – **Visualisation des filtres appris sur** *basic* **d'un RBM et des variantes RBM$_{update}$ et RBM$_{fprop}$, chacun employant une couche d'entrée de type *exponentielle tronquée*.** Les filtres montrés sont ceux qui permettaient de la meilleure performance de classification, soit avec une fraction ν de 0.25% et de 0.4% dans le cas de RBM$_{fprop}$ et RBM$_{update}$ respectivement. On remarque RBM$_{update}$ apprend des filtres intéressants, répondant à des patrons particuliers, tandis que ceux de RBM$_{fprop}$ ont très peu d'intérêt.

4.8.3 En résumé

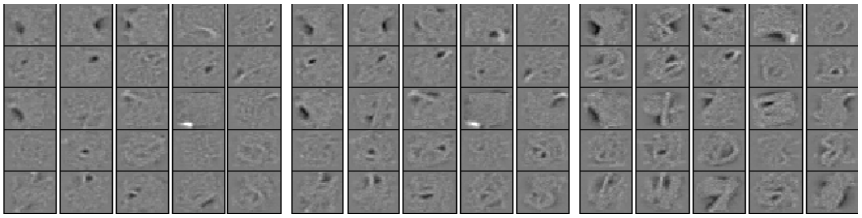
Parmi les nouvelles stratégies de pré-entraînement étudiées, nous avons vu que l'architecture SRAE avec bruit MN ou PS obtient la plus basse erreur de généralisation sur *basic* et *rot*, avec SDAE$_m n$ obtenant des performances équivalentes. Sur l'ensemble de données *bg-rand*, SMAE$_{OM}$ obtient la meilleure performance et SRAE$_{mn}$ performe de manière similaire. De plus, pour le cas du SMAE, une emphase sur la reconstruction des composantes étiquetées comme manquantes permettait une certaine diminution de l'erreur de généralisation, emphase dont le niveau utilisé devrait être exploré davantage.

4.9 Comparaison qualitative en tant que modèle générateur

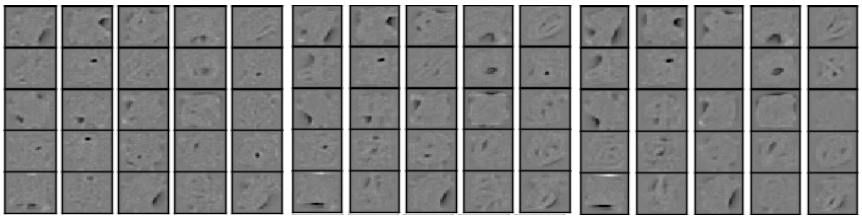
Bien qu'un DAE puisse être interprété comme un modèle génératif particulier, tel qu'expliqué par [45], la capacité des SAE et SDAE à générer des données n'avait pas encore été étudiée expérimentalement.

Nous avons voulu étudier leur capacité à générer des exemples variés tout comme celle d'une architecture à deux ou trois RBMs empilés. Afin d'y parvenir, nous avons utilisé un SDAE à trois couches cachées ayant été pré-entraîné seulement, mais donnant la meilleure performance de classification sur le jeu de données *basic* après avoir été entraîné avec le critère supervisé. Puis, nous avons exécuté le scénario de génération présenté à la figure 4.15, soit, une première phase d'encodage de l'image d'entrée vers une représentation cachée de premier, second ou troisième niveau, puis une phase d'alternance entre échantillonnage et décodage jusqu'à l'obtention de la génération d'un exemple. L'échantillonnage à partir d'un vecteur de reconstruction $z \in [0,1]^m$ est tiré d'une distribution de Bernoulli indépendante de moyenne z_i.

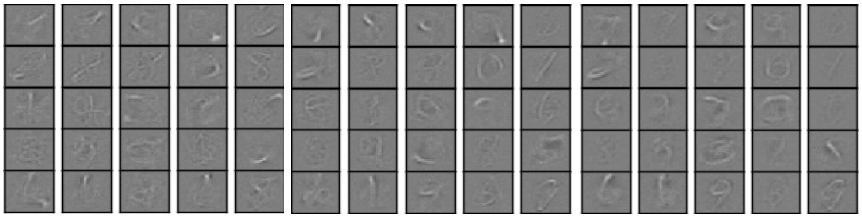
Bien que la génération à partir d'un DBN se fait habituellement de manière différente [21], pour fins de comparaison, nous avons utilisé la même procédure que la figure 4.15. Ceci afin de comparer le niveau de variabilité et la qualité de la génération, lorsqu'on utilise une représentation fixée au niveau supérieur, obtenue par une passe déterministe de bas en haut, et qu'on génère stochastiquement des exemples variés de haut en bas. Ainsi, on s'attendrait à ce que ces étapes répétées à quelques reprises, toujours



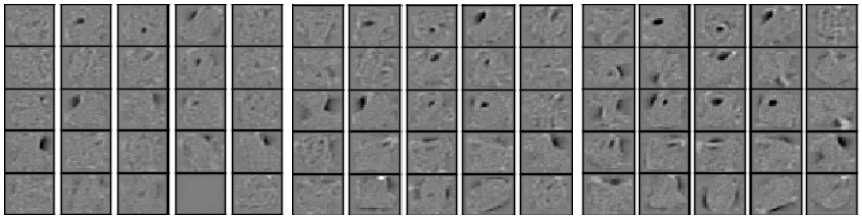
(a) *SDAE_mn*$\{10,25,50\}\%$



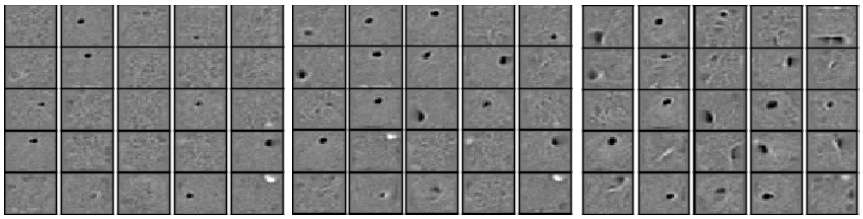
(b) *SRAE_mn*$\{10,25,40\}\%$



(c) *SRAE_ps*$\{10,25,40\}\%$



(d) $SMAE_{BM}\{10,25,40\}\%$



(e) $SMAE_{OM}\{10,25,40\}\%$

Figure 4.14 – **Filtres appris sur le jeu de données** *basic* **après pré-entraînement, avec différentes variantes de module de pré-entraînement :** *SDAE_mn*, *SRAE_*$\{mn,ps\}$ **et** $SMAE_{\{BM,OM\}}$**.** Pour chacun des modèles, trois groupes de filtres sont présentés, chacun étant le fruit d'un entraînement avec une certaine proportion de corruption appliquées à l'entrée. On observe que SRAE_{ps} apprend des filtres réagissant aux coups de crayons, contrairement aux autres modèles qui obtiendront des filtres s'activant plutôt pour une zone pleine de l'arrière plan.

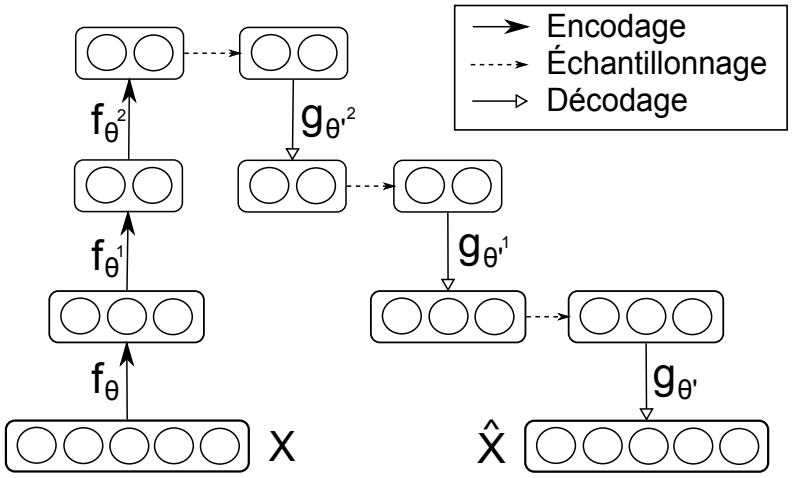
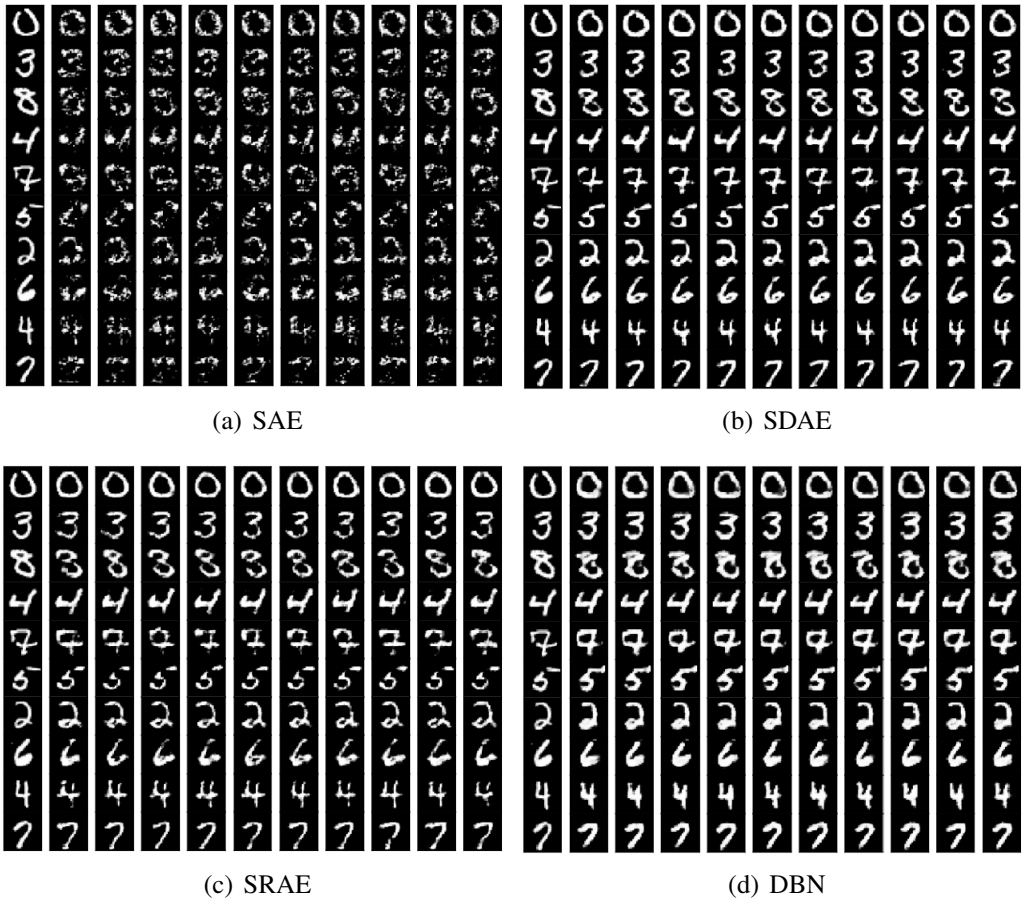


Figure 4.15 – **Scénario employé pour la génération d'images à partir d'une image d'origine.** Le scénario a été utilisé afin de vérifier la capacité des architectures DBN, SAE, SDAE et SRAE à générer des variantes d'une image d'origine. Il consiste en une première phase d'encodage, suivis d'une phase d'alternance entre échantillonnage et décodage. Ce scénario peut être effectué en considérant une seule, deux ou, comme dans le cas présenté, trois couches cachées.

à partir du même exemple d'entrée, permettent l'obtention en sortie d'une certaine variété de l'exemple d'entrée. La figure 4.16 compare les images générées via le scénario précédemment expliqué, des architectures SAE (4.16(a)), SDAE$_{ps}$ (4.16(b)), SRAE$_{ps}$ (4.16(c)) et finalement DBN (4.16(d)). On observe qu'effectivement, SAE s'avère être un piètre modèle génératif, tandis que SRAE, SDAE et DBN semblent tous trois être en mesure de générer des exemples de bonne qualité. Une différence notable est l'épaisseur accrue des chiffres générés par DBN, en comparaison avec ceux obtenus par SRAE et SDAE. Il en résulte possiblement une plus grande liberté de variétés pour ces deux derniers modèles, qui s'avèrent par exemple, capables de générer le chiffre 6 avec un trou, et ce, même si le chiffre 6 original est plein.

Par ailleurs, on s'attend à observer que l'augmentation du nombre de couches cachées impliquées dans le scénario vienne augmenter la variabilité retrouvée en sortie. La figure 4.17 confirme cette hypothèse pour le cas du modèle SDAE$_{ps}$. En effet, on remarque que l'usage d'une seule couche cachée résulte en la génération d'images très semblables à celle d'origine. Tandis que l'ajout d'une ou deux couches supplémentaires

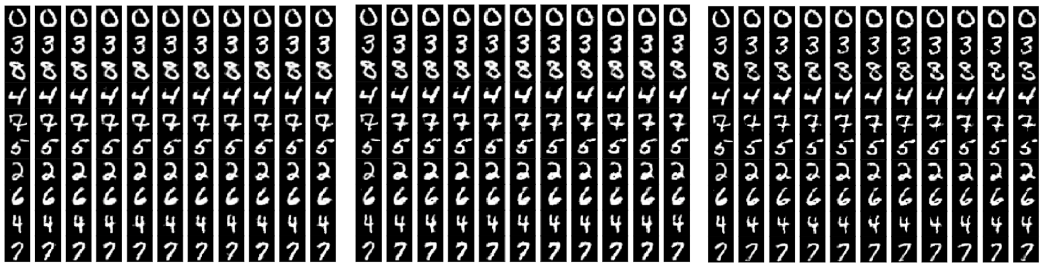


(a) SAE (b) SDAE

(c) SRAE (d) DBN

Figure 4.16 – **Comparaison de la capacité de génération de trois architectures à trois couches cachées : SAE (4.16(a)), SDAE avec bruit PS (4.16(b)), SRAE avec bruit PS (4.16(c)) et DBN (4.16(d)).** On remarque que l'architecture SAE n'est pas en mesure de générer de belles images en sorties. Tandis que SDAE, SRAE et DBN obtiennent un niveau de variété similaire pour le même scénario de génération.

permet l'obtention d'image avec une certaine variabilité observable. À remarquer par exemple, le cas du dernier chiffre 7, où la barre supérieure pour toutes les images générées par le SDAE à une couche est arrondie, tout comme l'originale, tandis qu'elle sera tantôt arrondie, tantôt droite dans le cas des images générées pour 2 et 3 couches. Ici aussi, il est possible d'observer que la génération d'un 6 avec un trou n'est obtenue qu'à partir de 2 couches.



(a) Une couche cachée (b) Deux couches cachées (c) Trois couches cachées

Figure 4.17 – **Comparaison du niveau de variété généré en fonction du nombre de couches cachées utilisées lors du scénario de génération.** On remarque que la variabilité s'accroît quelque peu avec l'augmentation du nombre de couches utilisées. En effet, on voit qu'avec l'usage d'une seule couche cachée, les images générées sont très proches de l'image originale, tandis qu'elles vont davantage différer avec deux et trois couches cachées.

4.10 Caractéristiques extraites utiles au SVM

Une architecture profonde pré-entraînée couche après couche selon un critère non-supervisé admet une meilleure performance de généralisation. Il est naturel de penser que cette procédure permet l'extraction de caractéristiques toujours de plus en plus abstraites de la couche d'entrée, jusqu'à l'obtention de caractéristiques plus pertinentes, facilitant la phase d'apprentissage selon un critère supervisé. Suivant cette logique, on s'attendrait à ce que les représentations ainsi apprises soient tout aussi utiles pour d'autres algorithmes d'apprentissage tels que les SVM. Par ailleurs, nous avons vu que les filtres de la première couche appris du SDAE étaient beaucoup plus intéressants que ceux du SAE, laissant présager des représentations davantage utiles au SVM. Cela devrait être d'autant plus vrai lorsque les représentations du SAE sont complètes ou sur-complètes puisqu'ils risquent alors d'apprendre des représentations inintéressantes liée à des solutions triviales au problème d'autoencodage. Ce sont ces hypothèses que nous avons voulu vérifier. Pour ce faire, nous avons sélectionné les SAE et SDAE$_{mn}$ composés de trois couches cachées de même taille, soit d'une taille supérieure à celle de la couche d'entrée, donnant la meilleure performance de généralisation sur les différents jeux de données et avons conservé ces modèles à l'état où ils avaient été pré-entraînés, sans entraînement

supervisé. La table 4.10 montre, pour chaque jeu de données, les hyper-paramètres du modèle SDAE$_{mn}$ ainsi retenu. A noter que nous indiquons par la même occasion le pas d'apprentissage de la phase supervisée offrant la meilleure performance à la classification. Après quoi, nous avons entraîné et testé un SVM linéaire puis gaussien, d'abord sur les données originales, puis sur les représentations issues de la première, deuxième et dernière couche cachée des modèles. Précisons ici que les performances que nous avons obtenues avec SVM gaussien sur les données originales diffèrent quelque peu de celles obtenues par [45], précédemment reportées dans la table 4.3, étant donnée des différences au niveau de la procédure d'exploration de l'espace des hyper-paramètres. Dans un premier temps, étudions la performance des SVM sur les représentations du SDAE. Les résultats concernés, situés dans la table 4.11, sont impressionnants. En effet, on remarque une nette amélioration de la performance de généralisation lorsque l'on utilise les représentations du SDAE et cette amélioration tend clairement à augmenter lorsque l'on utilise les représentations d'une couche de plus en plus haute dans l'architecture. Encore plus intéressant, les représentations sont telles qu'elles rendent les données davantage linéairement séparables de manière à ce que le SVM linéaire, malgré sa faible capacité, obtienne des performances se rapprochant de celles du SVM gaussien.

Pour une visualisation graphique de ces résultats obtenus sur deux des ensembles de données, soit *basic* et *rot-bg-img*, se référer à la figure 4.18. Pour le cas de *rot-bg-img* 4.18(b), deux courbes supplémentaires montrent la performance des SVM linéaire et gaussien lorsqu'ils utilisaient les représentations d'un réseau de neurones n'ayant subi aucun pré-entraînement, c'est-à-dire dont les paramètres ont été initialisés aléatoirement. On observe qu'avec l'usage des représentations de la troisième couche cachée du réseau de neurones aléatoirement initialisé, la performance décline quelque peu, pour le cas linéaire, et considérablement pour le cas gaussien.

Voyons maintenant l'effet de l'utilisation des représentations obtenues du SAE. Le tableau 4.12 révèle la comparaison entre l'efficacité des représentations de la troisième couche du SDAE versus celles du SAE de la même couche pour tous les jeux de données avec comme référence, la performance du SVM sur les données originales. On constate que les représentations du SAE, bien qu'elles améliorent aussi grandement la

performance du SVM, sont en général un peu moins efficaces que celles du SDAE. Nous avons obtenu la même conclusion pour les représentations de la première et deuxième couche cachée, résultats non montrés.

À la lumière de ces résultats, nous devons mettre en doute l'une de nos hypothèses : un pré-entraînement même sans implication de bruit (SAE) et avec représentations complètes ou sur-complètes, transforme malgré tout l'entrée en des représentations facilitant de plus en plus la tâche supervisée, plus l'on monte dans l'architecture. En effet, ces résultats viennent confronter ceux démontrés à la section précédente. Nous avions vu que dans le cas d'un SAE, il était impossible d'apprendre des filtres naturels se démar-

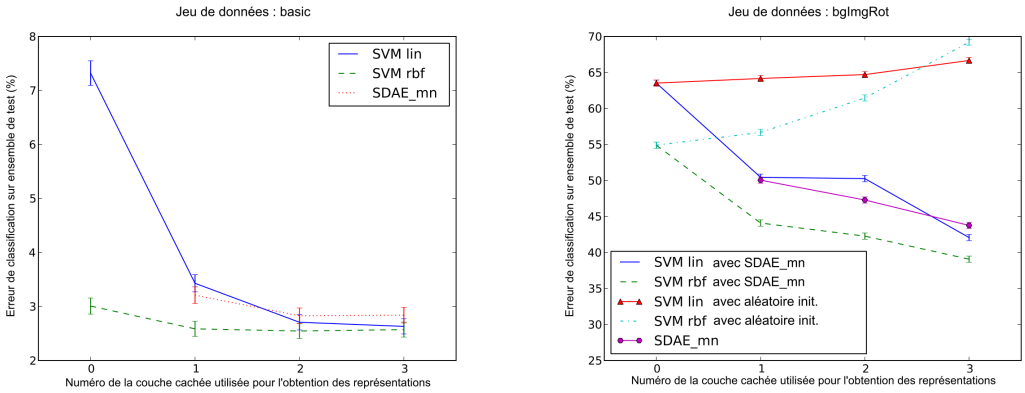
Tableau 4.10 – **Hyper-paramètres des modèles SDAE-3$_{mn}$ retenus pour l'extraction de caratéristiques.** Les réseaux ont trois couches cachées de même taille, soit une taille supérieure à celle de la couche d'entrée. Nous n'avons considérer que des couches cachées à 800 neurones pour *tzan-MPC* et 1000 pour tous les autres jeux de données. Les hyper-paramètres ont été sélectionnés de manière à obtenir une performance de classification satisfaisante, soit égale ou similaire à la meilleure performance jusqu'ici obtenue par un SDAE-3$_{mn}$ (dont le nombre d'unités cachées est, dans le cas de certains jeux de données, plus grand que 1000). C'est toutefois à l'état pré-entraîné seulement que les modèles ont été conservés pour l'obtention des représentations. Les hyper-paramètres sont, dans l'ordre, en commençant par la gauche : *(nHLay)* nombre de couches cachées, *(nHUnit)* nombre d'unités par couche cachée (même nombre pour chacune des couches), *(lRate)* pas d'apprentissage pour la phase non-supervisée, *(nEpoq)* nombre de passages sur l'ensemble d'entraînement pour la phase non-supervisée, *(lRateSup)* pas d'apprentissage pour la phase supervisée et *(fmi)* fraction de composantes d'entrée masquées à zéro.

Données	**nHLay**	**nHUnit**	**lRate**	**nEpoq**	**lRateSup**	**fmi**
MNIST	3	1000	0.005	10	0.1	25%
basic	3	1000	0.05	10	0.1	25%
rot	3	1000	0.005	100	0.1	10%
bg-rand	3	1000	0.005	125	0.005	25%
bg-img	3	1000	0.005	250	0.005	25%
rot-bg-img	3	1000	0.005	250	0.005	25%
rect	3	1000	0.05	100	0.005	10%
rect-img	3	1000	0.005	100	0.005	40%
convex	3	1000	0.0005	167	0.005	10%
tzan-MPC	3	800	0.0005	50	0.05	15%

quant tels que des filtres *à poids*, *à grilles* ou des filtres semblables à des filtres de Gabor.

Tableau 4.11 – **Comparaison des performances du SVM linéaire (lin) et gaussien (rbf) sur les données originales (SVM_0) et sur les représentations apprises au niveau de la première (SVM_1), deuxième (SVM_2) et dernière couche (SVM_3) d'un SDAE-3$_{mn}$ dont les couches cachées sont de taille égale (soit 1000 unités sauf pour *tzan-MPC* où 800 unités ont été utilisées.** Erreurs de test sur tous les jeux de données considérés, accompagnées d'un intervalle de confiance à 95%. La meilleure performance est en gras, de même que celles dont les intervalles de confiance se chevauchent. On voit clairement l'influence positive pour SVM linéaire comme gaussien de l'utilisation des caractéristiques extraites par le pré-entraînement non-supervisé du SDAE-3. De plus, la performance augmente avec l'utilisation de représentations plus haute dans l'architecture. Ainsi, SVM utilisant les représentations de la troisième couche cachée du SDAE offre les meilleures performances sur tous les jeux de données, battant ainsi les performances de SDAE.

Données	**SDAE-3**	**Type SVM**	**SVM_0**	**SVM_1**	**SVM_2**	**SVM_3**
MNIST	**1.28±0.22**	lin	5.33±0.44	2.12±0.28	**1.53±0.24**	**1.48±0.24**
		rbf	**1.61±0.25**	**1.26±0.22**	**1.39±0.23**	**1.38±0.23**
basic	3.01±0.15	lin	7.32±0.23	3.43±0.16	**2.71±0.14**	**2.63±0.14**
		rbf	3.03±0.15	**2.59±0.14**	**2.55±0.14**	**2.57±0.14**
rot	10.28±0.27	lin	43.47±0.43	21.74±0.36	15.15±0.31	10.00±0.26
		rbf	11.26±0.28	**8.45±0.24**	**8.27±0.24**	**8.64±0.25**
bg-rand	11.91±0.28	lin	24.14±0.38	13.58±0.30	13.00±0.29	11.32±0.28
		rbf	14.72±0.31	11.00±0.27	**10.08±0.26**	**10.16±0.26**
bg-img	15.12±0.31	lin	25.08±0.38	16.72±0.33	20.73±0.36	**14.55±0.31**
		rbf	22.65±0.37	15.91±0.32	16.36±0.32	**14.06±0.30**
rot-bg-img	43.76±0.43	lin	63.53±0.42	50.44±0.44	50.26±0.44	42.07±0.43
		rbf	54.90±0.44	44.09±0.44	42.28±0.43	**39.07±0.43**
rect	2.01±0.12	lin	29.04±0.40	6.43±0.22	2.31±0.13	1.80±0.12
		rbf	2.49±0.14	2.19±0.13	1.46±0.11	**1.22±0.10**
rect-img	22.16±0.36	lin	49.64±0.44	23.12±0.37	23.01±0.37	**21.43±0.36**
		rbf	23.18±0.37	22.27±0.36	21.56±0.36	**20.98±0.36**
convex	19.09±0.34	lin	45.75±0.44	24.10±0.37	18.40±0.34	**18.06±0.34**
		rbf	18.62±0.34	18.09±0.34	**17.39±0.33**	**17.53±0.33**
tzan-MPC	17.80±2.37	lin	20.72±2.51	12.51±2.05	7.95±1.68	**5.04±1.36**
		rbf	14.41±2.18	7.54±1.64	**5.20±1.38**	**4.13±1.23**



(a) (b)

Figure 4.18 – **Performance de classification des SVM linéaire et gaussien sur les images originales (numéro de la couche utilisée : 0) des jeux de données** *basic* **et** *rot-bg-img* **et sur les représentations obtenues d'un SDAE-3$_{mn}$, en comparaison avec la performance du SDAE-3$_{mn}$ lui-même.** Pour le cas de *rot-bg-img*, deux courbes supplémentaires montrent la performance des SVM lorsqu'entraînés sur des représentations issues d'un réseau de neurones initialisé aléatoirement (sans pré-entraînement). On voit très bien l'avantage de l'utilisation des représentations obtenues d'un SDAE-3$_{mn}$, tandis qu'un réseau sans pré-entraînement n'est pas en mesure d'extraire des caractéristiques particulièrement intéressantes à la tâche supervisée.

Si l'on prend comme exemple les filtres obtenus d'un SAE sur l'ensemble d'entraînement *basic*, ces derniers sont soit très locaux, soit ils se rapprochent qualitativement de ceux obtenus par une initialisation aléatoire. Tandis que dans le cas du SDAE, les filtres se personnalisent davantage et sont moins locaux (figure 2.4). Il semble donc que malgré l'apparence moins intéressante à première vue des filtres de la première couche d'un SAE, et malgré le fait que ce dernier ait l'opportunité d'apprendre une solution triviale peu intéressante, les filtres obtenus génèrent tout de même des représentations plus pertinentes que l'entrée brute elle-même pour l'apprentissage d'un SVM sur une tâche supervisée. Les matrices de connexions étant liées, il est possible que cela engendre une contrainte d'apprentissage empêchant le SAE d'apprendre la solution triviale, et donc de tendre plutôt vers une solution dont les représentations apprises s'en trouvent plus utiles à la tâche supervisée d'un SVM. Tandis que des matrices non-liées, pourraient offrir la liberté nécessaire au SAE d'apprendre cette solution triviale et ainsi d'apprendre des re-

présentations peu utiles à un SVM. Afin de vérifier cette nouvelle hypothèse, nous avons relancé les expériences sur *tzan-MPC* avec des matrices de connexion non-liées. Tel que nous révèle la table 4.13, malgré la capacité supplémentaire du modèle, les représentations apprises par le SAE avec matrices de poids non-liées améliorent toutes aussi bien sinon quelque peu mieux (quoi que la différence soit peu significative étant donnée le chevauchement des intervalles de confiance) la performance de généralisation des SVM gaussien et linéaire.

Une seconde façon d'expliquer le phénomène serait qu'étant donné que nous conservons le modèle SAE ayant seulement été pré-entraîné, mais dont les hyper-paramètres ont été sélectionnés en fonction de l'obtention de la meilleure performance de géné-

Tableau 4.12 – **Comparaison des performances de SVM linéaire et gaussien sur les représentations de la troisième couche cachée des architectures SDAE et SAE.** On remarquera que dans la majorité des cas, l'utilisation des représentations issues du SDAE offre de meilleures performances qu'avec l'utilisation des représentations du DAE.

Données	**Type SVM**	**SVM**$_0$	**SVM**$_{SDAE_3}$	**SVM**$_{SAE_3}$
basic	lin	7.32±0.23	**2.63±0.14**	3.26±0.16
	rbf	3.03±0.15	**2.57±0.14**	**2.80±0.14**
rot	lin	43.47±0.43	**10.00±0.26**	**9.87±0.26**
	rbf	11.26±0.28	**8.64±0.25**	9.42±0.26
bg-rand	lin	24.14±0.38	**11.32±0.28**	13.58±0.30
	rbf	14.72±0.31	**10.16±0.26**	12.80±0.29
bg-img	lin	25.08±0.38	**14.55±0.31**	21.53±0.36
	rbf	22.65±0.37	**14.06±0.30**	18.93±0.34
rot-bg-img	lin	63.53±0.42	**42.07±0.43**	47.03±0.44
	rbf	54.90±0.44	**39.07±0.43**	43.88±0.43
rect	lin	29.04±0.40	1.80±0.12	**1.35±0.10**
	rbf	2.49±0.14	**1.22±0.10**	**1.22±0.10**
rect-img	lin	49.64±0.44	**21.43±0.36**	22.56±0.37
	rbf	23.18±0.37	**20.98±0.36**	22.52±0.37
convex	lin	45.75±0.44	**18.06±0.34**	18.83±0.34
	rbf	18.62±0.34	**17.53±0.33**	18.45±0.34
tzan-MPC	lin	20.72±2.51	**5.04±1.36**	**4.89±1.34**
	rbf	14.41±2.18	**4.13±1.23**	**4.08±1.23**

ralisation suite à l'entraînement supervisé, le modèle prioriserait des hyper-paramètres d'entraînement non-supervisé aidant à la tâche supervisée au détriment de l'erreur de reconstruction du pré-entraînement. Une vérification a permis de constater que le nombre d'époques de même que le taux d'apprentissage utilisés pour l'entraînement non-supervisé pour la majorité des jeux de données sont assez faibles. Nous pourrions ainsi émettre l'hypothèse que le modèle SAE avec représentation complète ou sur-complète est porté à contraindre la tâche non-supervisée en utilisant un petit nombre d'époques et un pas de gradient insuffisant de sorte à éviter l'atteinte de la solution triviale. Il serait par ailleurs intéressant de pré-entraîner un SAE et d'utiliser, à chaque couche, l'arrêt prématuré afin de déterminer le nombre d'époques suffisant pour l'atteinte d'un minimum ou d'un plateau sur l'erreur de reconstruction. Il est à supposer que le modèle ainsi obtenu aura dès lors appris une solution triviale, que la performance de généralisation ne serait plus aussi bonne et que les représentations résultantes seraient moins utiles au SVM.

Tableau 4.13 – **Comparaison des performances des SVM linéaire et gaussien sur les représentations obtenues d'un SAE-3 avec matrices liées ou non-liées, sur le jeu de données** *tzan-MPC***.** Erreurs de test sur *tzan-MPC*, selon que le SAE-3 utilise des matrices de connexion liées ou non. Les résultats sont accompagnées d'un intervalle de confiance à 95%. Pour chacun des cas, la meilleure performance est en gras, de même que celles dont les intervalles de confiance se chevauchent. On observe que, contrairement à nos attentes, les SVM obtiennent une amélioration de performance lorsque les matrices sont non-liées, quoi que cette amélioration soit peu significative étant donné le chevauchement des intervalles de confiance.

Matrices de poids	**SAE-3**	**Type SVM**	**SVM$_1$**	**SVM$_2$**	**SVM$_3$**
SAE_Liées	16.94 ± 1.95	lin	12.38±2.04	**7.82±1.66**	**4.89±1.34**
		rbf	7.63±1.65	**5.22±1.38**	**4.08±1.23**
SAE_Non-liées	17.62 ± 1.91	lin	9.77±1.84	**5.81±1.45**	**4.89±1.34**
		rbf	**6.24±1.50**	**4.05±1.22**	**3.98±1.21**

CHAPITRE 5

CONCLUSION

Nos travaux ont débuté dans le contexte où d'abord, le pré-entraînement non-supervisé couche après couche d'une architecture profonde, procédure initiée par [22] avec les RBMs, avait fait ses preuves, initialisant les paramètres du réseau dans un bassin d'attraction plus favorable à la tâche supervisée. Puis, des études subséquentes avaient révélé que le remplacement des RBMs par des AEs permettait l'obtention de performances de classification presque aussi bonnes [5] et que l'ajout d'un processus de débruitage, soit l'utilisation du module DAE, éliminait cet écart de performance, voire même admettait des performances supérieures à celles obtenues par pré-entraînement de RBMs [45]. Par la même occasion, [45] avaient démontré le bénéfice sur la qualité des filtres appris, et conséquemment sur la qualité des représentations obtenues, d'un pré-entraînement par DAEs.

Notre recherche visait d'abord à appuyer ces précédentes études en validant l'utilité du pré-entraînement non-supervisé, et plus particulièrement celui issu de l'empilement de DAEs. L'objectif suivant était de se pencher sur le SDAE afin d'en faire ressortir d'intéressantes propriétés. Finalement, étant donné le bon fonctionnement du SDAE, nous avons voulu explorer l'usage de nouveaux modules de pré-entraînement, variantes des DAE ou des RBM, intégrant aussi un processus de bruitage.

Plus précisément, notre étude reproduit premièrement les résultats confirmant que le pré-entraînement permet l'entraînement efficace d'architectures de plus d'une couche cachée, et que le SDAE tire davantage profit de l'augmentation du nombre de neurones par couche que son prédécesseur, le SAE. En explorant l'architecture SDAE, nous sommes arrivés à la conclusion qu'un niveau de bruit non minimal est bénéfique, et que ce niveau de bruit se situe au sein d'un intervalle de valeurs assez large améliorant tout aussi considérablement la performance à la tâche supervisée. Ainsi, il n'est pas trop ardu pour un expérimentateur de trouver un niveau de bruit adéquat. Par ailleurs, l'essai des bruits PS et GS, mieux justifiés que MN, ont permis d'obtenir, en premier lieu,

de meilleures performances de classification. En second lieu, et ce fut là une découverte particulièrement intéressante, un DAE avec ce type de corruption entraîné sur des parties d'images naturelles, apprend des filtres d'allure semblable aux filtres de Gabor, que l'on retrouve également au niveau les cellules V1 du cortex visuel [25]. D'autres expériences ont confirmé que la procédure d'initialisation par débruitage était moins profitable à la tâche supervisée si on l'applique au niveau de la première couche seulement.

De plus la variante d'auto-encodeur débruiteur avec emphase sur la reconstruction de composantes corrompues (appliquée avec un bruit PS) s'est avérée prometteuse et mérite que l'on s'y penche davantage dans de futures études, notamment en explorant plus finement le niveau d'emphase.

Nous avons aussi pu démystifier certaines fausses croyances relatives à la notion d'entraînement avec bruit et de son lien avec le procédé d'entraînement au débruitage du SDAE. En premier lieu, nous avons montré empiriquement que l'entraînement d'un SDAE n'équivalait pas à la méthode d'entraînement sur des données pré-traitées par un processus de bruitage [23, 42] : un SVM gaussien ou un SAE, entraînés sur des données bruitées, paraissent loin d'être en mesure d'obtenir une performance de généralisation aussi bonne que le SDAE. En second lieu, nous avons vu empiriquement que l'entraînement sur des données avec un bruit Gaussien (par lequel nous avons pu, sur des images naturelles, apprendre des filtres de Gabor) n'est pas équivalent à l'utilisation d'une régularisation L2 sur les poids d'un auto-encodeur (avec quoi il nous a été impossible d'obtenir des filtres semblables à des Gabor). Ceci confirme que l'affirmation retrouvée dans [11] indiquant qu'un entraînement avec bruit GS est équivalent à l'application d'une régularisation de Tikhonov (équivalence par ailleurs valide uniquement dans la limite d'un bruit très petit), ne peut se traduire en l'équivalence d'un SDAE utilisant le bruit GS et d'un SAE entraîné avec régularisation L2.

En plus de nouveaux types de bruit MN et GS, et de la version avec emphase sur les composantes corrompues, nous avons investigué plusieurs autres variantes d'auto-encodeurs débruiteurs et de RBM avec introduction de bruit. Nous en avons évalué les performances au niveau de la tâche supervisée de même qu'inspecté visuellement les filtres appris. Les variantes étudiées sont : deux versions d'un auto-encodeur comblant

les données manquantes (SMAE$_{BM}$ et SMAE$_{OM}$), avec ou sans emphase sur la reconstruction des données manquantes, l'auto-encodeur rebruiteur (SRAE) puis la machine de Boltzmann restreinte faisant intervenir du bruitage à deux niveaux différents de son apprentissage (DBN$_{frop}$ et DBN$_{update}$). Nous avons établi le potentiel du SRAE, qui améliorait quelque peu les performances à la tâche de classification, et combiné à l'utilisation d'un bruit PS, apprenait sur *basic* des filtres détecteurs de traits de crayons. Du côté de l'architecture SMAE, SMAE$_{BM}$ performait mieux que SMAE$_{OM}$ sur deux des trois jeux de données et apprenait sur *basic* un mélange de filtres détecteurs de *gouttes* et de traits de crayons, tandis que SMAE$_{OM}$ obtenait la meilleure performance sur *bg-rand*et apprenait des filtres principalement détecteurs de *gouttes*. L'application d'une emphase sur les données manquantes, quant à elle, n'a révélé aucune amélioration significative de la performance de classification.

Par ailleurs nous avons montré que, contrairement au SAE, les paramètres du réseau profond appris par le SDAE lui permettent d'être utilisé de manière convaincante comme un modèle générateur. Une fois la couche supérieure fixée (par une passe déterministe de bas en haut, sorte d'inférence rapide à partir d'une image d'origine), la variété et la qualité des images générées par échantillonnage de haut en bas est semblable pour le SDAE et pour un DBN, alors qu'un SAE génère des images de très mauvaise qualité. Qui plus est, nous avons pu observer que la variabilité des images générées devenait plus importante et intéressante avec l'augmentation du nombre de couches utilisées lors du scénario de génération.

Une autre série de résultats particulièrement intéressants que nous avons obtenus sont ceux témoignant de l'efficacité d'extraction de caractéristiques pertinentes des architectures SAE et SDAE, résultant en des représentations utiles à l'apprentissage du SVM. Effectivement, nous avons vu que l'utilisation de ces représentations pour l'apprentissage d'un SVM linéaire comme gaussien permettait une baisse importante des erreurs de généralisation sur les neuf problèmes étudiés. Par ailleurs, cette baisse était d'autant plus notable, plus on utilisait des représentations hautes dans l'architecture. Aussi, nous avons été surpris de constater que les représentations du SAE étaient elles aussi pratiquement aussi utiles au SVM que celles du SDAE et ce, malgré l'observation de filtres

peu intéressants appris par un AE sur *basic* (figure 2.4) ou sur *olsh-12x12* (figure 4.6), et la piètre capacité générative des DAE. Ce point demeure un mystère.

En somme, les travaux du présent mémoire auront permis de décortiquer le phénomène de pré-entraînement via l'empilement de DAEs, de renforcer notre idée quant à ses avantages sur son prédécesseur, le SAE, notamment en comparant la qualité des représentations sur-complètes apprises à leur première couche, et d'ouvrir la porte à l'utilisation de nouveaux modules de pré-entraînement aussi performants, voire quelque peu meilleurs. Nous aurons conséquemment participé à une meilleure compréhension et au développement de nouvelles stratégies servant à l'initialisation d'architectures profondes pour la résolution de tâches supervisées. Il reste néanmoins beaucoup de questions sans réponse auxquelles de futures recherches tenteront d'apporter un plus ample éclairage. Notamment, cerner encore davantage les caractéristiques d'un algorithme qui rendent possible l'apprentissage de *bonnes* représentations, menant à une initialisation adéquate et à une meilleure performance à la tâche supervisée. Ceci aiderait grandement au développement de stratégies de pré-entraînement encore meilleures. Finalement, d'un point de vue pratique, il serait particulièrement avantageux de trouver un moyen de réduire le nombre d'hyper-paramètres impliqués dans l'entraînement d'un réseau de neurones profond car ceci simplifierait considérablement le pénible et coûteux processus de sélection du modèle.

BIBLIOGRAPHIE

[1] Mark A. Aizerman, Emmanuel M. Braverman, and Lev I. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25 :821–837, 1964.

[2] Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3) :643–674, 1996.

[3] Peter Auer, Mark Herbster, and Manfred K. Warmuth. Exponentially many local minima for single neurons. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8 (NIPS'95)*, pages 315–322. MIT Press, Cambridge, MA, 1996.

[4] E. B. Baum and D. Haussler. What size net gives valid generalization ? *Neural Computation*, 1 :151–160, 1989.

[5] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Adv. Neural Inf. Proc. Sys. 19*, pages 153–160, 2007.

[6] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, to appear, 2009.

[7] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007.

[8] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18 (NIPS'05)*, pages 107–114. MIT Press, Cambridge, MA, 2006.

[9] James Bergstra. Algorithms for classifying recorded music by genre. Master's thesis, Université de Montreal, 2006.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[11] Christopher M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1) :108–116, 1995.

[12] Miguel A. Carreira-Perpiñan and Geoffrey E. Hinton. On contrastive divergence learning. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, pages 33–40. Society for Artificial Intelligence and Statistics, 2005.

[13] Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In Yoshua Bengio, Dale Schuurmans, Christopher Williams, John Lafferty, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 342–350. NIPS Foundation, 2010 (to appear).

[14] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *AI & Stat.'2009*, 2009.

[15] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS'89)*, pages 524–532, Denver, CO, 1990. Morgan Kaufmann, San Mateo.

[16] Kenji Fukumizu and Shun-ichi Amari. Local minima and plateaus in hierarchical structures of multilayer perceptrons. *Neural Networks*, 13(3) :317–327, 2000.

[17] Patrick Gallinari, Yann LeCun, Sylvie Thiria, and Francoise Fogelman-Soulie. Memoires associatives distribuees. In *Proceedings of COGNITIVA 87*, Paris, La Villette, 1987.

[18] Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1 :113–129, 1991.

[19] Jay Hegdé and David C. Van Essen. Selectivity for complex shapes in primate visual area v2. *Journal of Neuroscience*, 20(5), March 2000. ISSN 1529-2401.

[20] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. Boltzmann machines : Constraint satisfaction networks that learn. Technical Report TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science, 1984.

[21] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. Boltzmann machines : Constraint satisfaction networks that learn. Technical Report TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science, 1984.

[22] Goeffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18 :1527–1554, 2006.

[23] Lasse Holmström and Petri Koistinen. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1) :24–38, 1992.

[24] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2 :359–366, 1989.

[25] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148 :574–591, 1959.

[26] Minami Ito and Hidehiko Komatsu. Representation of angles embedded within contour stimuli in area v2 of macaque monkeys. *Journal of Neuroscience*, 24(13) : 3313–3324, March 2004. doi : http://dx.doi.org/10.1523/JNEUROSCI.4364.

[27] Viren Jain and Sebastian H. Seung. Natural image denoising with convolutional networks. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Leon Bottou, editors, *Advances in Neural Information Processing Systems 21 (NIPS'08)*, 2008.

[28] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many

factors of variation. In Zoubin Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 473–480. ACM, 2007.

[29] Hugo Larochelle, Yoshua Bengio, Jerome Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10 :1–40, January 2009.

[30] Hugo Larochelle, Yoshua Bengio, Jerome Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10 :1–40, 2009.

[31] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient BackProp. In G. B. Orr and K.-R. Müller, editors, *Neural Networks : Tricks of the Trade*, pages 9–50. Springer, 1998.

[32] Yann LeCun. *Modèles connexionistes de l'apprentissage*. PhD thesis, Université de Paris VI, 1987.

[33] Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area V2. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 873–880. MIT Press, Cambridge, MA, 2008.

[34] Régis Lengellé and Thierry Denoeux. Training MLPs layer by layer using an objective function for internal representations. *Neural Networks*, 9 :83–97, 1996.

[35] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability, Vol. 1*, pages 281–296, 1967.

[36] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381 :607–609, 1996.

[37] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6) :559–572, 1901.

[38] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 1137–1144. MIT Press, 2007.

[39] Frank Rosenblatt. The perceptron — a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y., 1957.

[40] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323 :533–536, 1986.

[41] Sebastian H. Seung. Learning continuous attractors in recurrent networks. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10 (NIPS'97)*, pages 654–660. MIT Press, 1998.

[42] J. Sietsma and R. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1) :67–79, 1991.

[43] Paul Smolensky. Information processing in dynamical systems : Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, 1986.

[44] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1096–1103. ACM, 2008.

[45] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Int. Conf. Mach. Learn.*, pages 1096–1103, 2008.

[46] M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *NIPS 17*, Cambridge, MA, 2005. MIT Press.

[47] Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1168–1175, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi : http://doi.acm.org/10.1145/1390156.1390303.